Joonas Järvinen

Web-sovellus mainonnanhallintaprosessin työvaiheiden tueksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

23.4.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Joonas Järvinen Web-sovellus mainonnanhallintaprosessin työvaiheiden tueksi 41 sivua 23.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Teknologiajohtaja Maria Koskinen Yliopettaja Harri Airaksinen

Insinöörityö toteutettiin digimainonnan ja -medioiden kehittämiseen ja konsultointiin erikoistuneelle yritykselle. Insinöörityön tarkoituksena oli suunnitella ja toteuttaa web-sovellus, joka helpottaa mainonnanhallintaprosessin mainosinventaarin tarkistuksen ja mainoskampanjan varauksen työvaiheita. Web-sovellus koostuu kolmesta eri osasta: selainpuolesta, palvelinpuolesta ja tietokannasta. Lisäksi web-sovellus keskustelee Cxense Display -mainonnanhallintajärjestelmän kanssa ohjelmointirajapinnan välityksellä.

Web-sovelluksen selainpuoli toteutettiin käyttämällä HTML5-, CSS3- ja JavaScript-webtekniikoita. Selainpuolen käyttöliittymän toteutukseen käytettiin React-kirjastoa ja responsiivista React-Bootstrap-sovelluskehystä. Palvelinpuoli koostuu Express-sovelluskehystä hyödyntävästä Node.js-palvelimesta, johon luotiin API-päätepisteitä selainpuolta varten. Web-sovelluksen tietokantana käytettiin postgreSQL-tietokantaa.

Insinöörityön suunnitteluvaiheessa web-sovelluksen tärkeimmiksi asioiksi nousivat sovelluksen helppokäyttöisyys, turvallisuus ja laajentamismahdollisuus. Web-sovellukseen määriteltiin erikseen käyttöliittymä ja toiminnallisuudet tavallisille käyttäjille ja ylläpitäjille.

Insinöörityössä toteutettiin web-sovelluksen prototyyppi, jota testattiin kehitysympäristössä eri verkkoselaimilla ja päätelaitteilla. Web-sovelluksen toiminnallisuuksista ei löydetty testausvaiheessa puutteita, mutta sovellusta ei ole kuitenkaan viety tuotantoympäristöön ja sen tulevaisuus on vielä epävarma.

Avainsanat	web-sovellus, mainonnanhallinta, React, Node.js

Author	Joonas Järvinen
Title	Web application for supporting the ad operations workflow
Number of Pages	41 pages
Date	23 April 2017
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Maria Koskinen, Chief Technology Officer Harri Airaksinen, Principal Lecturer

This bachelor's thesis was made for a company that specializes in developing online advertising and media. The purpose of this bachelor's thesis was to design and develop a web application for supporting inventory checking and campaign booking phases of the advertising operations workflow. The web application consists of three parts which are the client-side, the server-side and a database. In addition, the web application communicates with external Cxense Display ad serving solution via application programming interface.

The client-side of the web application was developed using HTML5, CSS3 and JavaScript web technologies. The user interface of the client-side was developed using React JavaScript library and responsive React-Bootstrap framework. The server-side consists of a Node.js server with API endpoints. The Node.js server makes use of Express framework. PostgreSQL was used for the database of the web application.

In the design phase of this bachelor's thesis, ease of use, safety and a possibility for further development were chosen as the most important things for the web application. The functionalities of the web application were defined separately for common users and administrators.

A prototype of the web application was developed in this bachelor's thesis. The prototype was tested in the development environment with different browsers and devices. No lack of functionality was found in the testing process of the web application but the web application has not been deployed to production and the future of the application is uncertain at this time.

Keywords	web application, advertising operations, React, Node.js

Sisällys

Lyhenteet

IAB	Interactive Advertising Bureau. IAB on digimarkkinoinnin ja -mainonnan kasvua ja kehitystä ajava yhteisö.
API	Application programming interface. API eli ohjelmointirajapinta on määritelmä, jonka mukaan eri ohjelmat voivat keskustella keskenään.
SPA	Single-Page Application. Web-sovellus, joka toimii yksittäisellä sivulla luoden natiivisovellusta vastaavan käyttökokemuksen.
HTML	Hypertext Markup Language. HTML on avoimesti standardoitu hypertekstin merkintäkieli, jota käytetään erityisesti internetsivujen luomiseen.
CSS	Cascading Style Sheets. CSS on tyyliohjeiden laji, joka on suunnattu erityisesti internetsivujen ulkoasun määrittelyyn.
HTTP	Hypertext Transfer Protocol. HTTP on hypertekstin siirtoprotokolla, jota verkkoselaimet ja -palvelimet käyttävät tiedonsiirtoon.
JSON	JavaScript Object Notation. JSON on yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
XML	Extensible Markup Language. XML on rakenteellinen kuvauskieli, jota käytetään sekä tiedonvälityksen että dokumenttien tallentamisen formaattina.
AJAX	Asynchronous JavaScript And XML. AJAX tarkoittaa joukkoa tekniikoita, joilla JavaScriptillä toteutetaan asynkronisia HTTP-pyyntöjä, jotka palauttavaa tietoa JSON- tai XML-muodossa.
REST	Representational State Transfer. REST on HTTP-protokollaan perustuva ohjelmointirajapintojen toteuttamismalli.
JWT	JSON Web Token. JWT on JSON-pohjainen avoimen standardi tapa luoda käyttöoikeusvaltuuksia.

SQL	Structured Query Language. SQL on standardoitu kyselykieli, jolla relaatiotietokantaan voidaan tehdä hakuja, muutoksia ja lisäyksiä.
URL	Uniform Resource Locator. URL on merkkijono, jota käytetään osoittamaan internetsivua.

1 Johdanto

Insinöörityön tarkoituksena on saada aikaan web-sovellus, joka helpottaa digitaalisen mainonnanhallinnan prosessia. Digitalisoitumisen vuoksi mainostajat ovat siirtyneet laajenevissa määrin käyttämään digitaalisia medioita perinteisten medioiden sijaan. Digimainonta onkin nykyään yksi isoimmista tulonlähteistä julkaisijoille ja mainonnanhallinta yksi sen keskeisimmistä prosesseista.

Insinöörityön toimeksiantaja on digimainonnan ja -medioiden kehittämiseen ja konsultointiin erikoistunut Relevant Partner 4 Media Oy. Relevant auttaa asiakkaitaan parantamaan mainonnan tuottoja, kasvattamaan yleisön arvoa ja luomaan uusia tulovirtoja tarjoamalla muun muassa digimainonnan operatiivisia palveluita (AdOps), konsultointia ja koulutuksia. Relevant on kansainvälinen yritys, joka toimii Helsingissä ja Kööpenhaminassa.

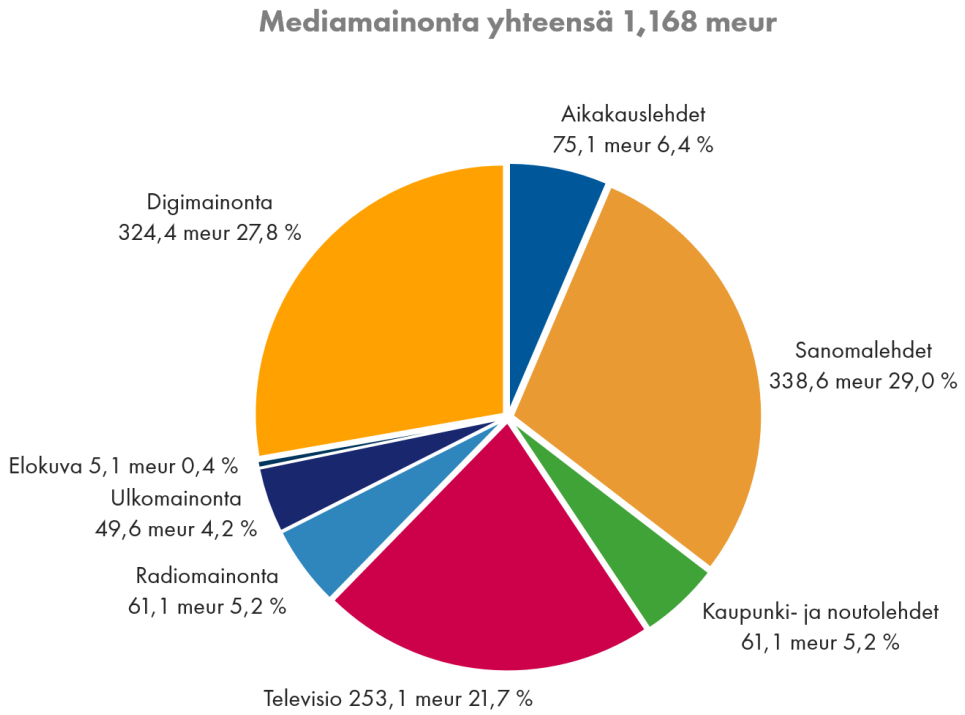
Tässä insinöörityöraportissa käsitellään web-sovelluksen kehitysprosessia suunnittelusta valmiin prototyypin toteutukseen. Suunnitteluvaiheessa määritellään web-sovelluksen käyttötarkoitus, toiminnallisuudet ja käyttöliittymä. Suunnitteluvaiheessa hahmotellaan myös web-sovelluksen kokonaisarkkitehtuuri ja otetaan huomioon turvallisuus. Lisäksi raportissa tarkastellaan web-sovelluksessa käytettäviä selain- ja palvelinpuolen tekniikoita esimerkkejä hyödyntäen.

Toteutusvaiheessa käydään läpi kehitysympäristön rakentaminen, tietokannan käyttö ja sekä selain- että palvelinpuolen kehitysprosessit. Selain- ja palvelinpuolen kehitysprosessissa käydään läpi yhden toiminnallisuuden toteuttaminen yksityiskohtaisesti koodiesimerkkien avulla ja muut toiminnot pääpiirteittäin.

2 Digimainonta ja -mainonnanhallinta Suomessa

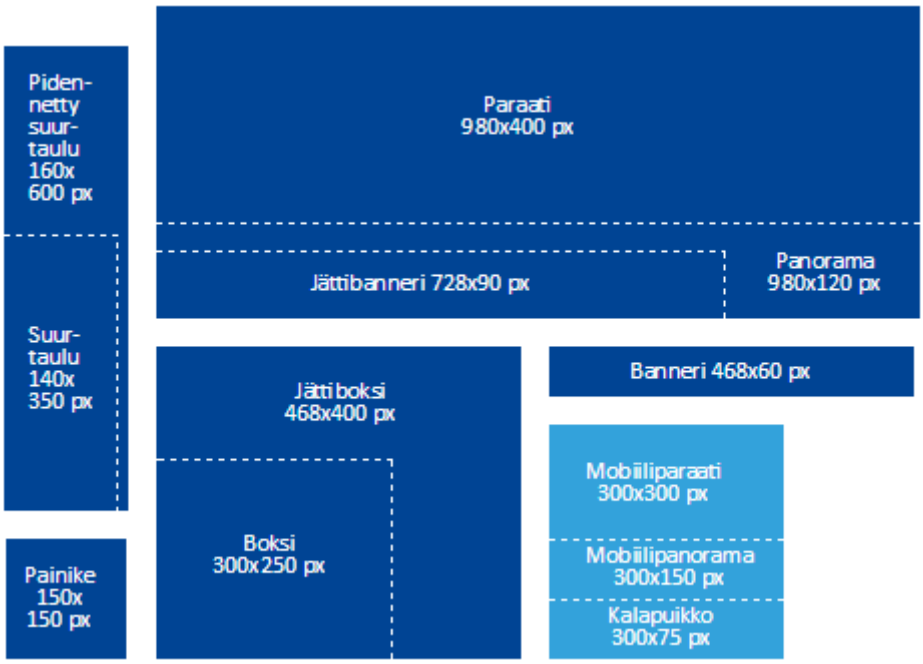
Lähes jokaisen digimedian, eli internetissä toimivan median, yksi tärkeimmistä tulonlähteistä on mainonta. Jokainen internetin käyttäjä on törmännyt eri internetsivuilla oleviin erikokoisiin ja eri paikoissa oleviin mainoksiin. Jokaisella kerralla, kun joku näkee internetsivulla olevan mainoksen, tai joissain tapauksissa klikkaa sellaista, se tietää kyseiselle digimedialle rahaa. Digimedian käyttäjät muodostavat median mainosinventaarin, jota mediat sitten myyvät mainostajille.

Suomen digimainonta on ollut viime vuosina kovassa kasvussa. Vuonna 2016 digimainonnan kasvu lähes kaksinkertaistui, 12,6 prosenttiin, vuoden 2015 6,8 prosentista. Vuoden 2016 jälkeen digimainonnan osuus koko Suomen mediamainonnasta on 27,8 prosenttia eli jo lähes kolmannes (kuva 1). IAB Finlandin (Interactive Advertising Bureau) mukaan digimainonta jatkaa kovaa kasvuaan. (1.)



Kuva 1. Suomen mediamainonnan panostukset vuonna 2016 (1).

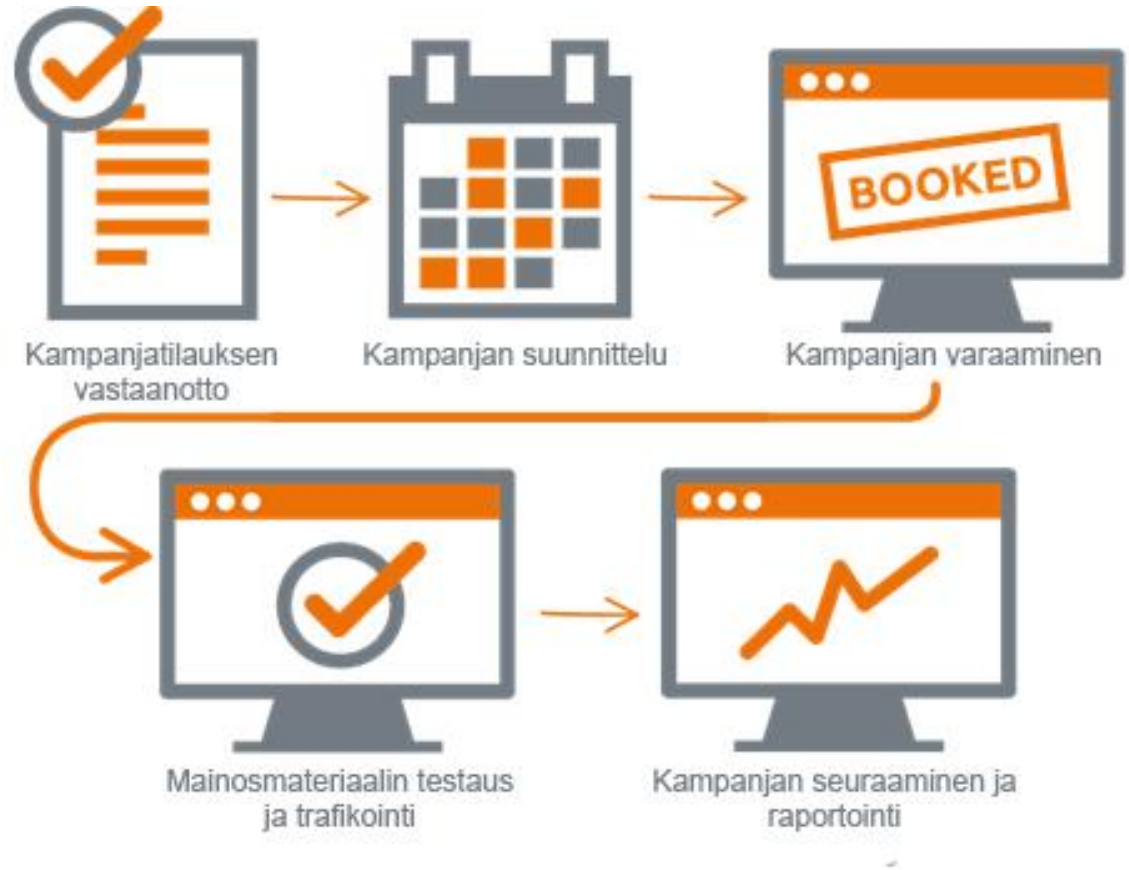
Erikokoisia mainoksia kutsutaan mainosmuodoiksi tai -formaateiksi. Isoimmat mainosmuodot on yleensä sijoitettu internetsivujen yläosiin korkeamman huomioarvon saavuttamiseksi. Mainosmuodot ovat eri hintaisia mainostajalle, ja ison huomioarvon mainosmuodot ovat kalliimpia. Jotta mainostajat voisivat vaivattomasti käyttää samoja mainoksiaan useissa eri medioissa, on mainosmuotoja varten laadittu yhteisiä standardeja ja suosituksia (kuva 2). IAB Finland julkaisee vuosittain uusia digimainonnan standardeja ja suosituksia. IAB Finland on Suomen digimarkkinoinnin ja -mainonnan kasvua ja kehitystä ajava yhteisö. (2; 3.)



Kuva 2. IAB Finlandin perusmainosmuotojen suositukset (2).

Yhden mainostajan tiettynä ajanjaksona tietylle kohdeyleisölle näkyviä mainoksia kutsutaan mainoskampanjaksi. Mainoskampanjalla on usein myös muun muassa näyttötavoite, eli digimedia on myynyt mainostajalle tietyn määrän mainosnäyttöjä, mikä tarkoittaa sitä määrää, kuinka monta kertaa mainos näytetään median vierailijoille. Jotta mainoskampanja saadaan kokonaisuudessaan toteutettua, se vaatii mainostilan myyntityön lisäksi mainonnanhallinnan tehtäviä. (4.)

Mainonnanhallinnan tehtäviin kuuluu muun muassa digimedian mainosinventaarin tarkistus ja varaaminen, mainostajan mainosmateriaalin tarkastus ja trafikointi (ad trafficking) sekä mainoskampanjan seuraaminen ja raportointi (kuva 3). Digimedioilla on lähes poikkeuksetta käytössä mainonnanhallintajärjestelmä, joka on integroitu suoraan niiden internetsivustojen yhteyteen. Mainonnanhallinnan tehtävät suoritetaan mainonnanhallintajärjestelmää käyttäen.



Kuva 3. Mainonnanhallinnan työprosessi (5).

Mainosinventaarin tarkistuksella tarkoitetaan sitä, että tarkistetaan digimedian mainosinventaarin määrä esimerkiksi tietyltä ajanjaksolta tietyillä kohdennuksilla. Kohdennus voi olla esimerkiksi maantieteellinen sijainti tai päätelaite. Mainostaja voi esimerkiksi haluta kohdentaa mainoksena näkymään Helsingissä oleville Android-älypuhelinten käyttäjille ja näyttää mainoksiaan kyseiselle kohderyhmälle 100 000 kertaa viikon aikana. Jos mainosinventaaria on tarpeeksi vapaana, voidaan mainosinventaari varata kyseiselle mainostajalle. (6.)

Mainosmateriaalin tarkastuksessa mainostajalta saadun tiedoston, joka voi olla esimerkiksi kuvatiedosto, ominaisuudet tarkistetaan. Kuvatiedostosta tarkistetaan tyypillisesti dimensiot eli mitat ja tiedoston koko, jotta ne ovat ennalta määrättyjen ohjeiden mukaiset. Mainosmateriaaleille on digimedian toimesta määritelty muun muassa enimmäiskoko, koska internetsivulla näytettävät mainokset vaikuttavat suoraan sivun lataamisnopeuteen. Jos mainosmateriaali on ohjeistuksen mukainen, se trafikoidaan mainonnanhallintajärjestelmään. Trafikoimisella tarkoitetaan sitä, että mainosmateriaali asetetaan mainonnanhallintajärjestelmään luotuun mainoskampanjaan ennalta sovittu-

jen asetusten mukaisesti. Asetuksia ovat esimerkiksi ajanjakso, jolla mainosmateriaalia tulee näyttää kävijöille, ja näyttömäärä. (7.)

Mainoskampanjan ollessa käynnissä sen etenemistä seurataan. Mainonnanhallintajärjestelmät osaavat ennustaa, kuinka monta mainosnäyttöä kukin kampanja on saamassa kampanja-aikanaan. Jos ennusteen mukaan mainoskampanjan näyttömäärä on jäämässä sovittua näyttötavoitetta alhaisemmaksi, voidaan mainoskampanjaan tehdä muutoksia sen ollessa käynnissä, jotta näyttötavoite saavutettaisiin. Mainonnanhallintajärjestelmien ennuste perustuu internetsivuilta kerättyyn dataan, eli käytännössä mitattuihin kävijämääriin. Mitä enemmän dataa mainonnanhallintajärjestelmällä on sivustojen kävijämääristä, sitä tarkemmin se osaa ennustaa tulevaisuuden mainosnäyttöjen määriä. (8.)

Mainoskampanjan päätyttyä suoritetaan raportointi. Mainonnanhallintajärjestelmän mainoskampanjoista voidaan luoda raportteja, joista selviää hyvinkin yksityiskohtaisia tietoja mainoskampanjan suorituksesta. Näitä tietoja on muun muassa kokonaisnäyttömäärän jakautuminen kampanja-ajan eri päiville tai jopa tunneille, mainoksia klikanneiden määrä ja päätelaitteet, joissa mainos on näkynyt. (9.)

3 Web-sovelluksen määrittely

3.1 Käyttötarkoitus

Tässä insinöörityössä tehtävän web-sovelluksen käyttötarkoitus on helpottaa mainonnanhallintaprosessin mainosinventaarin tarkistuksen ja mainoskampanjan varauksen työvaiheita. Lähtötilanteessa mainosinventaarin tarkistus ja mainoskampanjan varaus suoritetaan Cxense Display -mainonnanhallintajärjestelmällä (https://wp.cxense.com/solutions/cxense-display/) (10). Cxense Display -mainonnanhallintajärjestelmän kyseiset työvaiheet ovat kömpelöitä, eikä järjestelmä ole lainkaan responsiivinen, joten sen käyttö on hyvin hankalaa pieninäyttöisillä päätelaitteilla. Näiden työvaiheiden helpottamiseksi suunniteltiin web-sovellus, joka keskustelee Cxense Display -mainonnanhallintajärjestelmän ohjelmointirajapinnan (API) kanssa erillisestä käyttöliittymästä, joka on helppokäyttöisempi ja responsiivinen.

Web-sovellus suunnitellaan siten, että sen käyttötarkoitus on tarvittaessa laajennettavissa kattamaan myös mainonnanhallintaprosessin muita työvaiheita, kuten kampanjaseurantaa.

3.2 Käyttöliittymä ja toiminnot

Sovelluksen käyttäjät jaettiin käyttäjiin ja ylläpitäjiin, joille määrättiin oikeudet eri toimintojen suorittamiseen ja eri näkymien tarkasteluun.

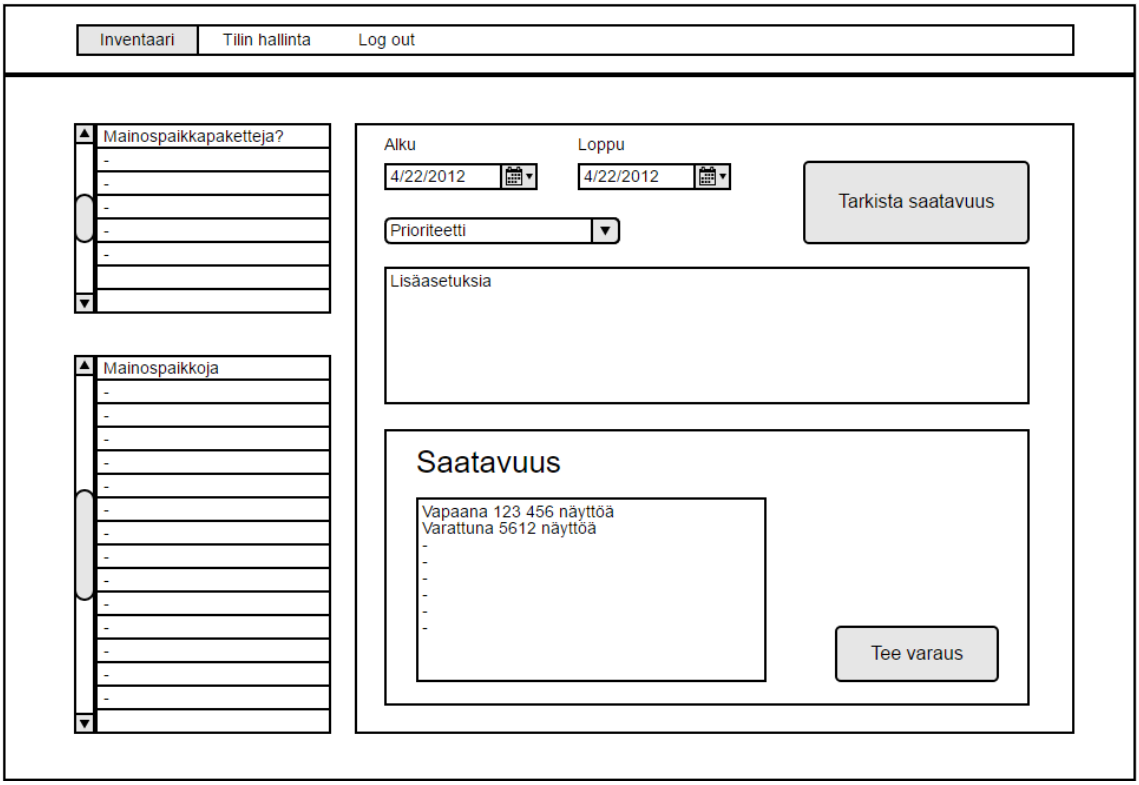
Käyttäjien tulee pystyä suorittamaan seuraavat toiminnot:

- sisäänkirjaus
- uloskirjautus
- mainosinventaarin tarkistus
- mainoskampanjan varaus
- salasanan vaihto.

Ylläpitäjien tulee pystyä suorittamaan kaikki samat toiminnot kuin käyttäjienkin, mutta lisäksi myös seuraavat toiminnot:

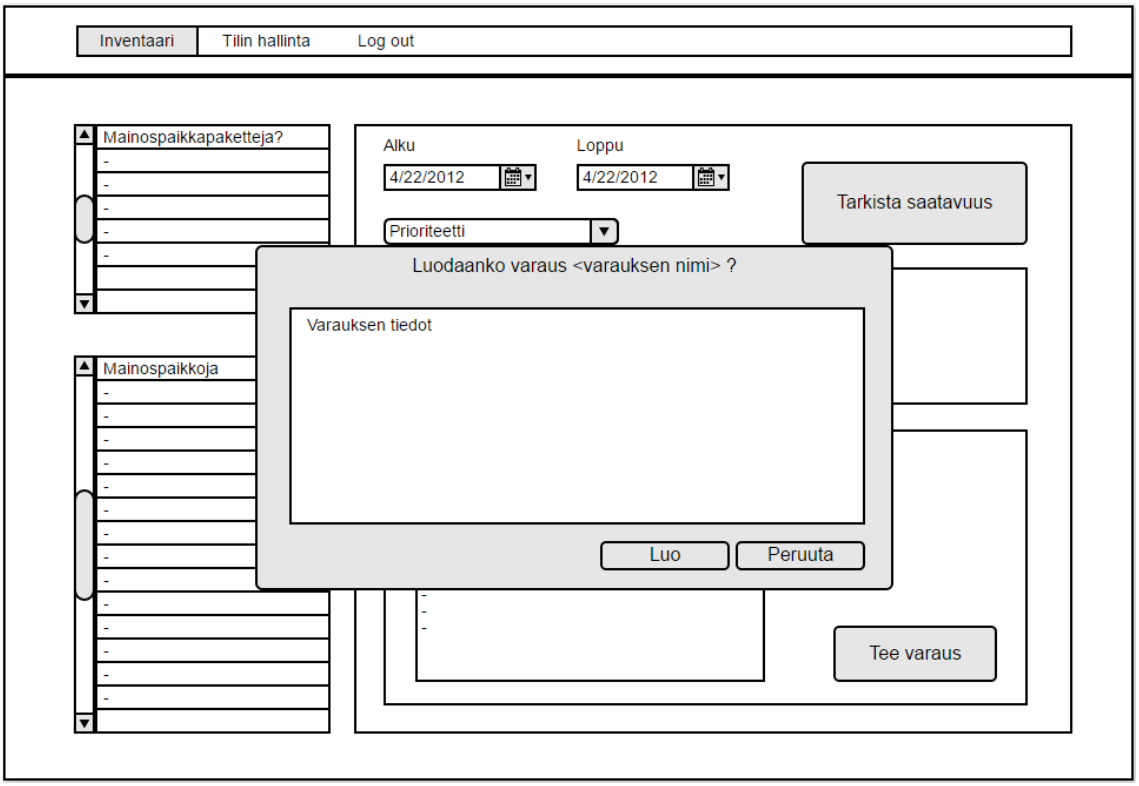
- käyttäjän lisäys
- käyttäjän muokkaus
- tapahtumalokien tarkastelu.

Käyttöliittymä koostuu sovelluksen ensimmäisessä versiossa kahdesta eri päänäkymästä, jotka ovat mainosinventaarintarkastusnäkymä ja käyttäjähallintanäkymä. Päänäkymien välillä liikutaan yläreunaan sijoitetun navigaatiopalkin avulla (kuva 4).



Kuva 4. Luonnos mainosinventaarinäkymästä sisäänkirjautumisen jälkeen.

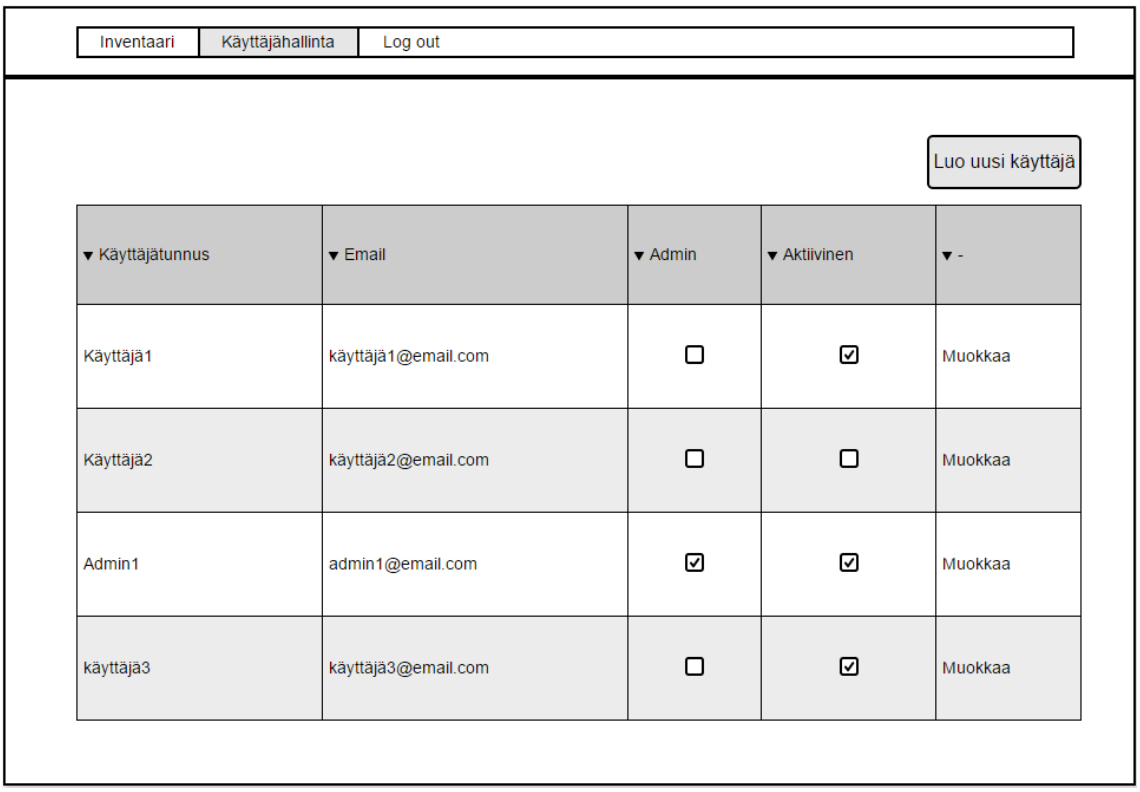
Mainosinventaarinäkymässä käyttäjä pystyy valitsemaan mainospaikkoja, alku- ja loppupäivämäärän, prioriteettitason ja mahdollisia lisäasetuksia, joiden perusteella mainosinventaaria voidaan tarkistaa (kuva 4). Mainosinventaarin tarkistamisen jälkeen on mahdollista varata mainoskampanja samoja asetuksia hyödyntäen. Mainoskampanjan luonti vahvistetaan erikseen mainosinventaarinäkymän päälle aukeavassa modaalissa (kuva 5).



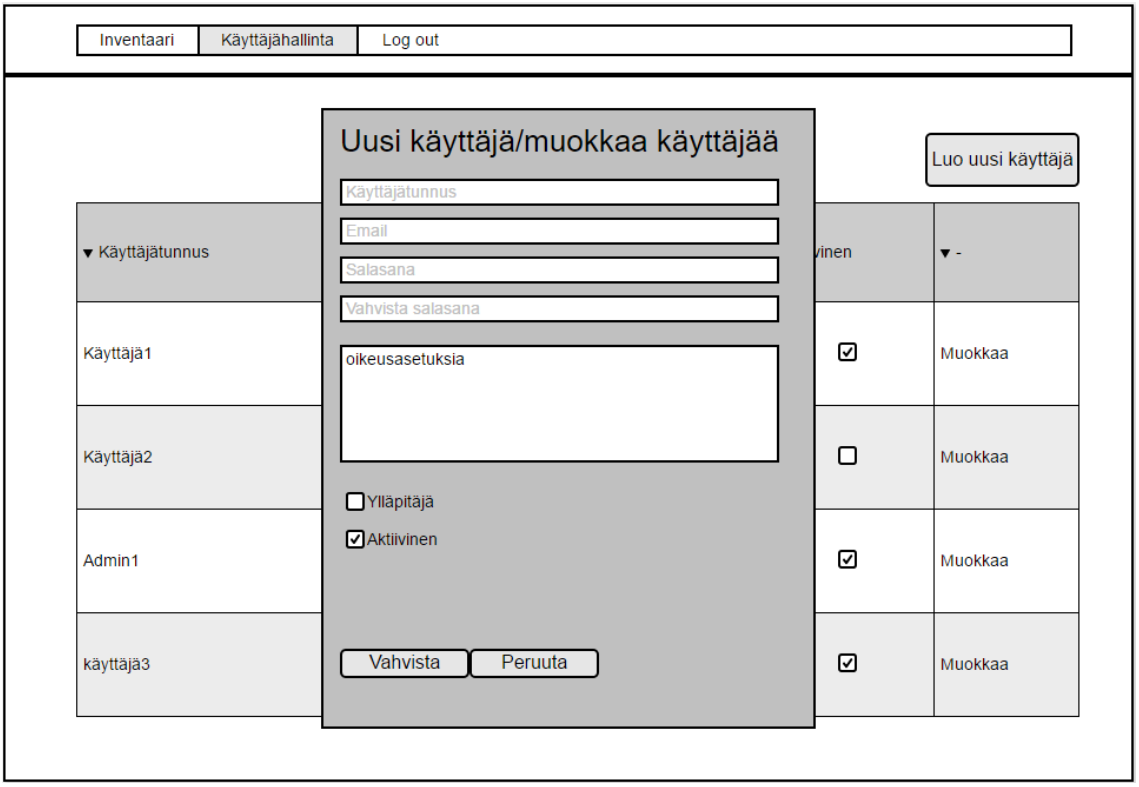
Kuva 5. Luonnos mainoskampanjan varaamisesta mainosinventaarinäkymässä.

Käyttäjähallintanäkymässä ylläpitäjän on mahdollista luoda uusia käyttäjiä ja muokata olemassa olevia käyttäjiä. Käyttäjähallintanäkymässä listataan sovellukseen luodut käyttäjätunnukset ja käyttäjien sähköpostiosoitteet. Listauksesta käy myös ilmi, onko kyseessä ylläpitäjä ja onko käyttäjätunnus aktiivinen (kuva 6). Ylläpitäjän luodessa tai muokatessa käyttäjää käyttäjähallintanäkymän päälle avautuu modaali, johon käyttäjän tietoja voidaan syöttää ja tämän jälkeen vahvistaa tai peruuttaa muutokset (kuva 7).

Käyttöliittymän yläreunan navigaatiopalkki ei ole samanlainen käyttäjille ja ylläpitäjille, vaan se mukautuu sisäänkirjautumisen jälkeen käyttäjän oikeuksien mukaisesti.



Kuva 6. Luonnos ylläpitäjän käyttäjähallintanäkymästä.

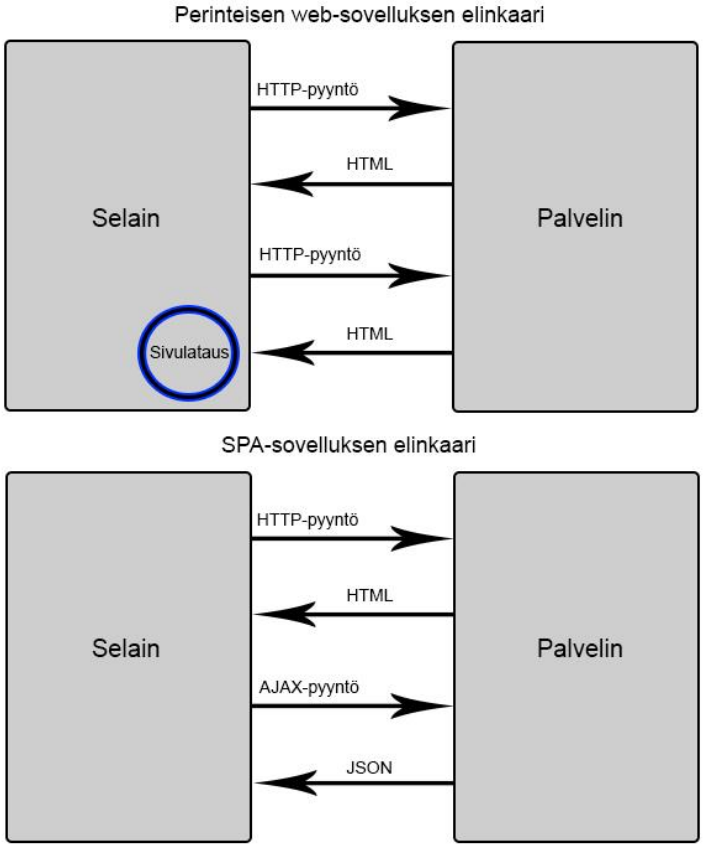


Kuva 7. Luonnos käyttäjän luomisesta ja muokkaamisesta käyttäjähallintanäkymässä.

4 Web-sovelluksessa käytettävät tekniikat

4.1 SPA-arkkitehtuuri

Web-sovelluksen arkkitehtuuriksi valittiin SPA-arkkitehtuuri (Single-Page Application). Siinä missä perinteisien web-sovelluksien rakenne muodostuu useista HTML-sivuista, joita ladataan käyttäjän siirtyessä sivulta toiselle, SPA-sovelluksissa sivulatauksia ei tapahdu (kuva 8). (11; 12.)



Kuva 8. Perinteisen web-sovelluksen ja SPA-sovelluksen elinkaaret.

Kuten kuvassa 8 näkyy, perinteisien web-sovelluksien elinkaari perustuu siihen, että käyttäjän verkkoselain tekee HTTP-pyyntöjä (Hypertext Transfer Protocol) sovelluksen palvelimelle, joka palauttaa aina HTML-sivun. Tämä taas aiheuttaa joka HTTP-pyynnön yhteydessä sivulatauksen.

Kuvan 8 mukaisesti SPA-sovellukset muokkaavat käyttöliittymää dynaamisesti AJAX-pyyntöjä (Asynchronous JavaScript and XML) hyödyntäen käyttäjän navigoidessa so-

velluksessa. Palvelin vastaa AJAX-pyyntöihin esimerkiksi JSON-muodossa (JavaScript Object Notation) olevalla datalla, jonka perusteella käyttöliittymää voidaan päivittää ilman erillistä sivulatausta. Tämä tekee sovelluksen käytöstä huomattavasti sujuvampaa ja käyttäjäystävällisempää. SPA-sovellukset ovat myös kevyempiä käyttää, koska koko sivua ei tarvitse ladata aina uudestaan, vaan ainoastaan oleelliset kohdat. (11; 12.)

SPA-sovelluksissa selainpuoli ja palvelinpuoli on selkeästi erotettu toisistaan. Selain- ja palvelinpuoli keskustelevat usein HTTP-protokollaan perustuvan REST-rajapinnan (Representational State Transfer) välityksellä. REST-rajapinta mahdollistaa tarvittaessa sovelluksen helpon laajentamisen. REST-rajapinnan ansiosta esimerkiksi eri alustoilla toimivan sovellukset voivat hyödyntää samaa palvelinpuolta. (11; 12.)

SPA-arkkitehtuurissa on kuitenkin myös huonot puolensa, joihin lukeutuu muun muassa hakukoneoptimoinnin ja sivuhistorian hallinnan haastavuus. Tässä web-sovelluksessa nämä ominaisuudet eivät kuitenkaan ole oleellisia, joten SPA-arkkitehtuurin valinta ei tuottanut ongelmia. (11; 12.)

4.2 Selainpuolen tekniikat

Web-sovelluksissa käytettävät tekniikat ovat olleet viime vuosina suuressa murroksessa. Nykypäivänä web-sovelluksien oletetaan toimivan useilla eri päätelaitteilla käyttöliittymästä ja näytön resoluutiosta riippumatta. Niin sanotut modernit web-tekniikat, joita tässäkin web-sovelluksessa käytetään, ovat yleistyneet ja kehittyneet vastaamaan näitä nykypäivän haasteita. Moderneihin web-tekniikoihin lukeutuvat HTML5, CSS3 ja JavaScript sekä niiden erilaiset kirjastot ja sovelluskehykset. Seuraavassa käydään tarkemmin läpi tämän web-sovelluksen selainpuolella käytettyjä tekniikoita.

HTML5

HTML (Hypertext Markup Language) on merkintäkieli, josta kaikki verkkosivut muodostuvat, kuten myös tämän web-sovelluksen selainpuoli. HTML muodostuu sisäkkäisistä ja peräkkäisistä elementeistä, jotka määrittelevät verkkosivun rakenteen. HTML ei kuitenkaan tuo toiminnallisuutta sivustolle, vaan ainoastaan sisällön. HTML:n viidennen version, eli HTML5:n, myötä elementtien määrä kasvoi, ja HTML5 mahdollistaa muun muassa videoelementin määrittelemisen. (13; 14.)

HTML5 ei ole pelkästään HTML-merkintäkielen viides versio, vaan se mahdollistaa myös uusien CSS3-tyylimääritteiden ja JavaScript-rajapintojen hyödyntämisen. Merkittävimpiä HTML5:n ominaisuuksia ovat

- uudet semanttiset elementit, kuten header-, footer-, article- ja section-elementit, jotka kuvaavat verkkosivun sisältöä tarkemmin
- svg- ja canvas-elementit, jotka mahdollistavat grafiikan piirtämisen; myös esimerkiksi 3D-grafiikan esittäminen on mahdollista
- audio- ja videoelementit, jotka mahdollistavat multimedian toistamisen ilman erillistä selainlaajennusta, kuten esimerkiksi Adoben Flash-soitinta
- uudet rajapinnat, jotka mahdollistavat muuan muassa käyttäjän sijainnin tai kameran hyödyntämisen käyttäjän luvalla
- useat suorituskykyä parantavat uudistukset, jotka parantavat käyttäjäkokemusta. (14; 15.)

CSS3

CSS (Cascading Style Sheets) on eräs tyyliohjeiden laji, jota käytetään erityisesti verkkosivustojen ulkoasujen määrittelyyn. Myös tämän web-sovelluksen ulkoasu määriteltiin CSS:llä. CSS-tyyliohjeilla voidaan esimerkiksi muuttaa tekstin väriä, fonttia ja kokoa. CSS-tyyliohjeita voi määritellä suoraan HTML-dokumenttiin käyttämällä HTML:n style-elementtiä tai viittaamalla ulkoiseen css-päätteiseen tiedostoon. (16.)

JavaScript

JavaScript on verkkoselaimen tulkitsema kevyt ohjelmointikieli, jota lähes kaikki verkkosivustot käyttävät toiminnallisuuksien luomiseen. JavaScript on dynaaminen prototyyppeihin perustuva kieli, joka tukee myös imperatiivista ja funktionaalista ohjelmointia sekä olio-ohjelmointia. JavaScript on tapahtumapohjainen kieli, eli verkkosivuston käyttäjän tekemiin toimintoihin, esimerkiksi painikkeen klikkaukseen, voidaan liittää JavaScriptin avulla toiminnallisuutta. (17; 18.)

JavaScript perustuu ECMAscript-standardiin. ECMAscriptin versio 5.1. julkaistiin vuonna 2012, ja kaikki modernit selaimet tukevat tätä standardia. ECMAscriptin versio 6 (ES2015) julkaistiin vuonna 2015, mutta kaikki modernit selaimet eivät vielä täysin tue sitä. Tämän web-sovelluksen kehitysvaiheessa hyödynnettiin ECMAscript 6:n ominai-

suuksia, mutta ne muunnetaan automaattisesti ECMAscript 5:n tukemaan muotoon ennen tuotantovaiheeseen menoa Babel-nimisen JavaScript-kääntäjän avulla. (17; 19.)

Vaikka JavaScript on tunnettu nimenomaan verkkosivujen toiminnallisuuksien luomisesta, sitä voidaan käyttää myös verkkoselaimen ulkopuolella, kuten palvelinympäristöissä. Tässä web-sovelluksessa JavaScriptiä käytetään sekä selain- että palvelinpuolen toiminnallisuuksissa. (18.)

JavaScriptin käytön helpottamiseksi on vuosien aikana kehitetty runsaasti JavaScript-kirjastoja. Tietotekniikassa kirjastoilla tarkoitetaan kokoelmia luokkia, ohjelmia ja aliohjelmia, jotka on tarkoitettu ohjelmoinnin helpottamiseen. Ne ratkaisevat yleensä yleisiä ongelmia mahdollisimman tehokkaasti, ja ohjelmoijat voivatkin aina tarpeen tullen lisätä tiettyjä kirjastoja sovelluksen eri osa-alueille. Tässä web-sovelluksessa käytetään muun muassa React-nimistä JavaScript-kirjastoa, josta kerrotaan tarkemmin seuraavassa. (20.)

React

React on Facebookin kehittämä JavaScript-kirjasto käyttöliittymän kehitykseen. Facebook kehitti Reactin alun perin omaan käyttöönsä, ja se käyttääkin sitä isoimmissa palveluissaan Facebookissa ja Instagramissa. Viime vuosina Reactin suosio on kasvanut räjähdysmäisesti, ja monet muutkin isot palvelut, kuten suoratoistopalvelu Netflix, käyttävät sitä palveluissaan. (21; 22.)

React on komponenttipohjainen JavaScript-kirjasto. Reactilla käyttöliittymän osat voidaan pilkkoa pieniin osiin, eli komponentteihin, joita voidaan mahdollisuuksien mukaan uudelleen käyttää sovelluksessa tai jopa useissa sovelluksissa. Tämä tekee käyttöliittymän kehityksestä tehokasta, koska samoja rakennuspalikoita voidaan hyödyntää moneen kertaan. (22; 23.)

Kuvan 9 esimerkissä web-sovellus on muodostettu viidestä eri React-komponentista. Esimerkissä on havainnollistettu uudelleenkäytettävyys tuotekategoriarivi- ja tuoterivikomponenttien avulla, joita esiintyy esimerkissä useita.

Kuva 9. Esimerkkisovellus, jossa eri React-komponentit on korostettu väreillä (23).

Reactia käytettäessä voidaan hyödyntää sen omaa JSX-syntaksia (koodiesimerkki 1), joka yhdistelee JavaScriptiä ja HTML-merkintäkieltä. JSX-syntaksi helpottaa React-ohjelmointia, mutta sen käyttö ei ole kuitenkaan pakollista. Reactia voidaan kirjoittaa myös puhtaalla JavaScriptillä. (21.)

```
class HelloMessage extends React.Component {
      render() {
            return <div>Hello {this.props.name}</div>
      }
}

ReactDOM.render(<HelloMessage name="John" />), mountNode)
```

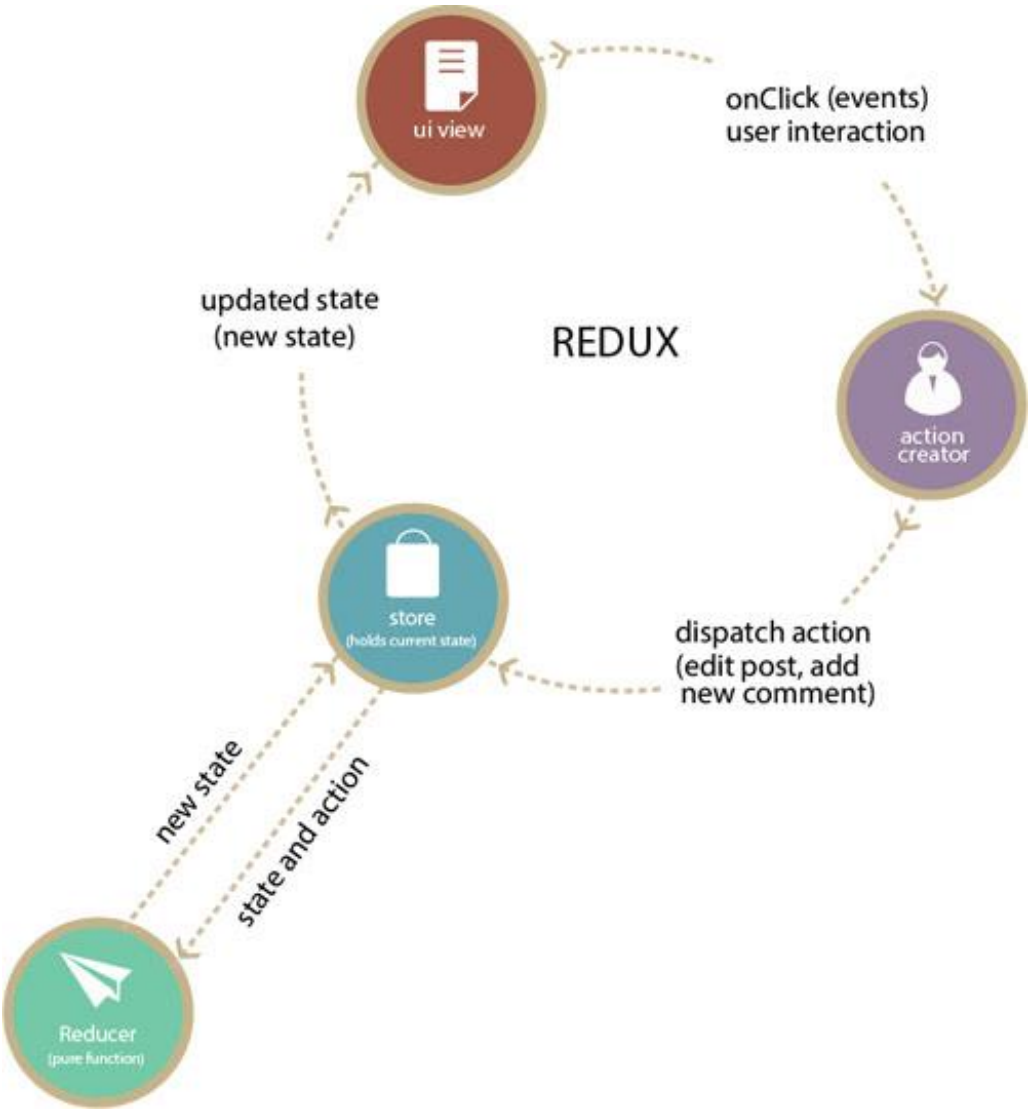
Koodiesimerkki 1. JSX-syntaksilla kirjoitettu React-komponentti (21).

Redux

Redux on JavaScript-kirjasto, jota käytetään web-sovelluksen tilanhallintaan. Tässä yhteydessä web-sovelluksen tilalla tarkoitetaan esimerkiksi sitä, mitä tietoa käyttöliittymässä juuri sillä hetkellä näytetään. Reduxia voidaan pitää eräänlaisena web-sovelluksen selainpuolen data-arkkitehtuurina. Redux kehitettiin vuonna 2015, ja se on noussut nopeasti isoon suosioon sen yksinkertaisuuden ja kattavan dokumentaation takia. Reduxia käytetään useimmiten erityisesti React-kirjaston kanssa web-sovellusten

käyttöliittymän kehityksessä. Myös tässä web-sovelluksessa Redux valittiin käytettäväksi selainpuolen tilanhallintaan sen yhteensopivuuden React-kirjaston ja yksinkertaisuuden takia. Google Chrome -verkkoselaimeen on saatavilla myös Reduxille räätälöityjä web-kehitystyökaluja, jotka helpottavat kehitystyötä ja ongelmanratkaisua. (24; 25.)

Reduxin toiminta perustuu siihen, että se säilyttää koko web-sovelluksen tilaa yksittäisessä muuttumattomassa tila-objektissa (store). Reduxin säilyttämää tilaa ei voi koskaan suoraan muuttaa, vaan muutoksen tapahtuessa silloisen tilan pohjalta luodaan uusi tila-objekti. Tämä lähestymistapa ennaltaehkäisee muun muassa web-sovelluksen virhetilanteita, joita saattaa esiintyä helpommin sellaisten tilanhallintaan käytettävien JavaScript-kirjastojen kanssa, jotka muokkaavat suoraan web-sovelluksen tilaa. Kuvassa 10 esitetään Reduxin tilanhallintaprosessi. (24; 25.)



Kuva 10. Reduxin tilanhallintaprosessi (26).

Kuvassa tapahtumaketju lähtee liikkeelle käyttäjän interaktiosta, joka on tässä tapauksessa klikkaus. Käyttäjän interaktion tapahtumakäsittelijä lähettää toiminnon, joka voi olla esimerkiksi kommentin lisäys, ja sovelluksen aikaisemman tilan Reducerille. Reducer on pelkkä puhdas funktio, joka vastaanottaa toiminnon ja aikaisemman tilan argumentteina ja palauttaa sovelluksen uuden tilan. Store säilyttää sovelluksen uutta tilaa, jonka perusteella React-komponentit päivittyvät automaattisesti, mikä taas näkyy käyttäjälle käyttöliittymässä tapahtuvana muutoksena, esimerkiksi hänen kirjoittamansa kommentin lisäyksellä. (24; 25; 26.)

HTML-sovelluskehys

Nykypäivänä web-sovellusten responsiivisuus on lähes itsestäänselvyys. Responsiivisuudella tarkoitetaan sitä, että web-sovellus mukautuu päätelaitteen näytön mukaisesti ja että sovelluksen käyttö on yhtä sulavaa päätelaitteesta riippumatta. Tämän haasteen ratkaisemisen helpottamiseksi on luotu lukuisia HTML-sovelluskehyksiä. (27.)

HTML-sovelluskehys koostuu valmiista paketista, joka sisältää usein HTML-, CSS- ja JavaScript-tiedostoja. Nämä tiedostot muodostavat sovelluskehyksen pohjan ja pitävät sisällään mahdollisesti valmiita käyttöliittymäelementtejä, esimerkiksi painikkeita ja lomakkeita. Tämä helpottaa web-sovelluksen kehittämistä, koska kehitystyötä ei tarvitse lähteä tekemään tyhjästä, vaan kehittäjät voivat hyödyntää sovelluskehyksen tarjoamia ominaisuuksia ja räätälöidä niitä haluamallaan tavalla. Nykypäivän HTML-sovelluskehykset ovat myös lähes poikkeuksetta responsiivisia ja tukevat suosituimpia verkkoselaimia, jolloin kehittäjien ei tarvitse rakentaa käyttöliittymää erikseen yhteensopivaksi eri selaimille. (27.)

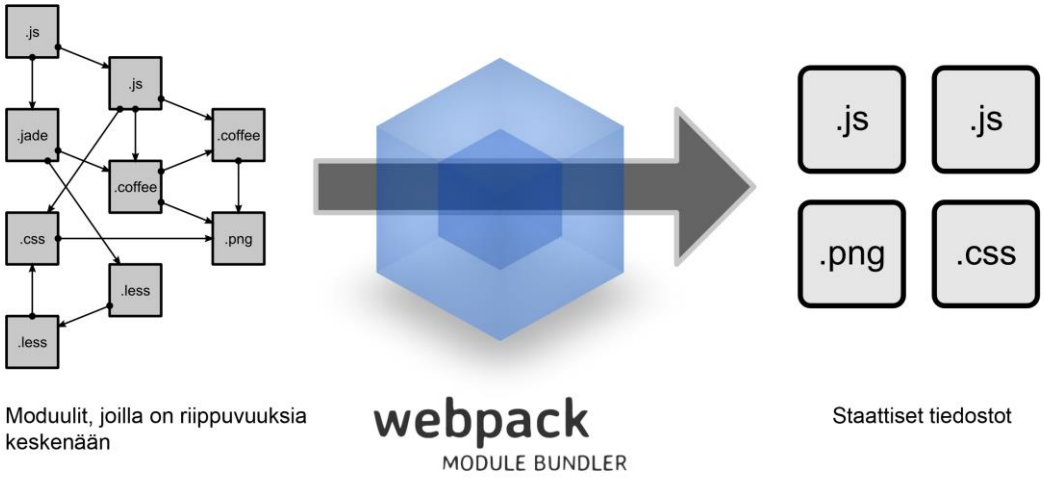
Tässä web-sovelluksessa käytettiin React-Bootstrap-sovelluskehystä, joka on Bootstrap-sovelluskehyksestä Reactin JSX-syntaksille räätälöity sovelluskehys (28).

Webpack

Web-sovelluksen kehitysvaiheessa on usein käytössä esimerkiksi useita erilaisia JavaScript-kirjastoja, HTML-sovelluskehys, CSS-tyyliohjeita ja useita muita tiedostoja, jotka yhdessä muodostavat lopullisen sovelluksen. Näillä tiedostoilla tai tiedostorakenteilla, eli moduuleilla, on usein riippuvuuksia toisiin tiedostoihin. Esimerkiksi jossain JavaScriptillä kirjoitetussa React-komponentissa voidaan viitata CSS:llä kirjoitettuun tyyliohjeeseen, jossa taas on määritelty jonkin HTML-elementin taustakuvaksi jokin

kuvatiedosto. Lisäksi sovelluksen kehitysvaiheessa voidaan haluta ohjelmoida JavaScriptin uusimmalla ES2015-versiolla, mutta muuntaa kirjoitettu JavaScript-koodi vanhempaan versioon ennen tuotantovaihetta, jotta kaikki yleisimmät selaimet varmasti tukisivat kirjoitettua koodia. JavaScriptin ES2015-versiota hyödynnettiin myös tämän web-sovelluksen kehitysvaiheessa. (29; 30.)

Jotta kehitysvaiheen moduulit ja kirjoitetut koodit saataisiin helposti haluttuun lopulliseen muotoonsa ennen tuotantovaihetta, voidaan hyödyntää olemassa olevia moduuliniputtajia (module bundler). Suosituimpia moduuliniputtajia ovat muun muassa Webpack (kuva 11), Browserify, Grunt ja Gulp, joista ensimmäistä käytetään tässä websovelluksessa.



Kuva 11. Webpack-moduuliniputtaja (30).

Kun Webpack otetaan käyttöön web-sovelluskehityksessä, se tulee ensin konfiguroida. Konfiguroinnissa määritellään muun muassa, miten Webpackin halutaan käsittelevän tietyntyyppisiä tiedostoja, kuten esimerkiksi JavaScript-, CSS- tai kuvatiedostoja. Webpack konfiguroidaan JavaScriptillä. (29; 31.)

Koodiesimerkin 2 riveillä 3–6 määritellään polku työstettävän web-sovellukseen. Riveillä 7–11 määritellään polku sijaintiin, johon Webpack kokoaa staattiset tiedostot niputtamisen jälkeen. Riveillä 13–17 määritellään, että Webpack käsittelee kaikki js-päätteiset tiedostot Babel-nimisellä JavaScript-kääntäjällä. Tämä mahdollistaa JavaScriptin kirjoittamisen uusimmalla ES2015-versiolla sovelluksen kehitysvaiheessa. Babel-

kääntäjä muuntaa kirjoitetun JavaScriptin vanhempaan versioon, jota selaimet ymmärtävät.

```
const webpackConfig = {
   devtool: 'source-map',
   entry: [
      'webpack-hot-middleware/client',
      './client/index.js'
   ],
   output: {
      filename: '[name].[hash].js',
      path: path.join(__dirname, 'dist'),
      publicPath: '/'
   },
   module: {
      loaders: [{
         test: /\.(js|jsx)$/,
         exclude: /node_modules/,
         loaders: ['react-hot', 'babel']
      }, {
         test: /\.(scss|css)$/,
         loaders: [
            'style-loader',
            'css-loader',
            'autoprefixer?browsers=last 2 version',
            'sass-loader'
         ]
      }]
   }
};
```

Koodiesimerkki 2. Osa tämän web-sovelluksen Webpackin konfigurointitiedostosta.

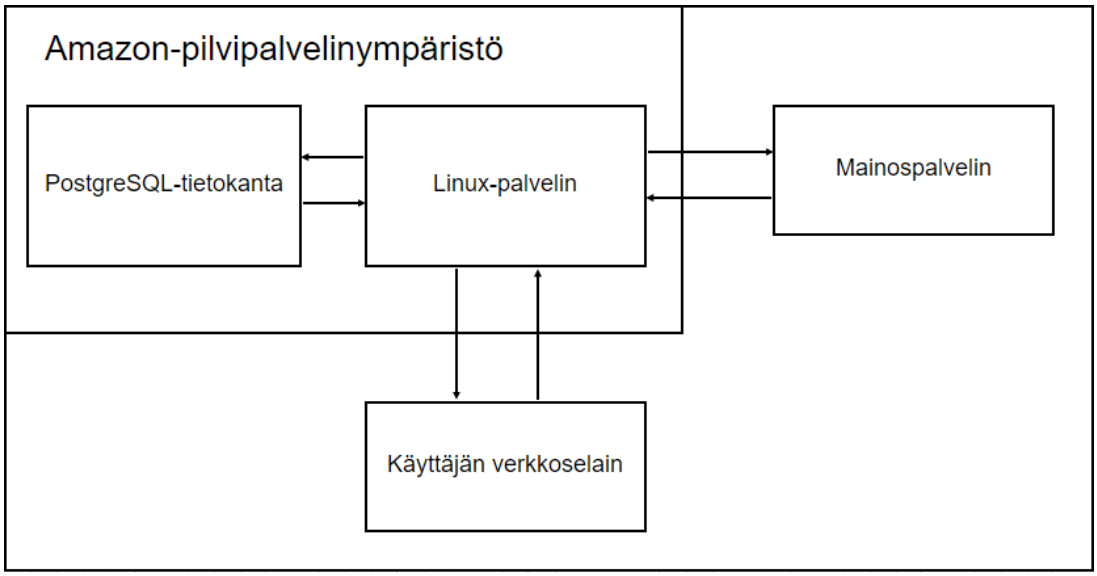
Webpackin hyötyjä ovat muun muassa

- turhan koodin sivuuttaminen: jos web-sovelluksessa viitataan esimerkiksi isoon CSS-tyyliohjeeseen, Webpack osaa karsia sieltä mukaan vain ne tyylimääritteet, joita sovellus oikeasti käyttää
- eri tiedostotyyppien käsittely halutulla tavalla
- erilaisten Webpack-lisäosien hyödyntäminen kehitysvaiheessa: ne tekevät ohjelmoinnista helpompaa; esimerkiksi React-hot-loader-lisäosa päivittää kehitysympäristön JavaScriptin ilman erillistä sivulatausta, kun JavaScript-tiedostoihin tehdään muutoksia (32; 34; 35).

4.3 Palvelinpuolen tekniikat

Palvelin tarkoittaa tietotekniikassa tietokoneessa suoritettavaa palvelinohjelmistoa ja tätä ohjelmistoa suorittavaa tietokonetta. Palvelimen tehtävänä on tarjota toimintoja ja palveluja sitä hyödyntävälle sovellukselle, eli asiakkaalle. Tässä insinöörityössä asiakkaalla tarkoitetaan sovelluksen käyttäjän verkkoselainta. (33.)

Tämän web-sovelluksen palvelinympäristönä käytettiin Amazonin pilvipalvelinympäristöä, johon oli asennettu Node.js-suoritusympäristö Linux-palvelimelle ja PostgreSQL-relaatiotietokantapalvelin erilliselle palvelimelle (kuva 12). Tässä web-sovelluksessa käytettäviä palvelinpuolen tekniikoita käsitellään tarkemmin seuraavassa.



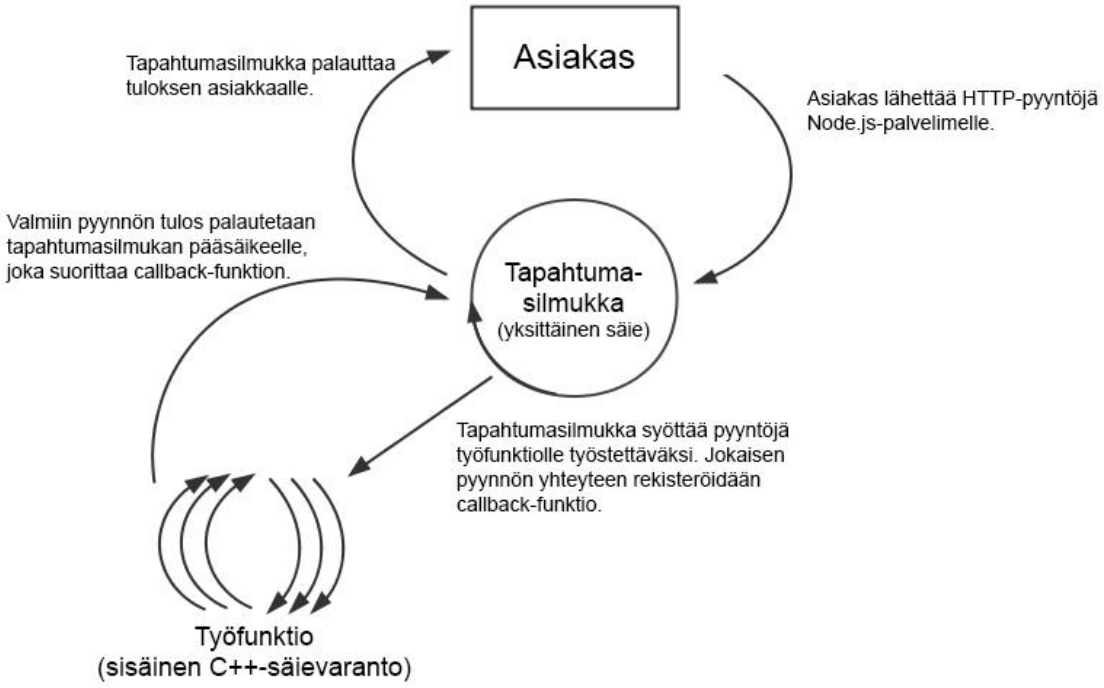
Kuva 12. Hahmotelma insinöörityön web-sovelluksen kokonaisrakenteesta.

Node.js

Node.js on avoimen lähdekoodin palvelinpuolen JavaScript-suoritusympäristö, joka hyödyntää Googlen kehittämää V8 JavaScript-moottoria JavaScriptin tulkkaamisen. Node.js on tapahtumapohjainen ja esteetön, eli mikään käsky ei estä toisen käskyn suorittamista. Esteettömyys tekee Node.js-suoritusympäristöstä kevyen ja tehokkaan, joten se soveltuu hyvin etenkin skaalautuvien web-sovellusten rakentamiseen. (34; 35.)

Node.js toimii yhden pääsäikeen varassa, mikä eroaa perinteisistä verkkopalvelintekniikoista, joissa jokainen yhteys saa oman säikeen. Node.js-palvelimen vastaanottaes-

sa HTTP-pyyntöjä asiakkaalta sen tapahtumasilmukka käynnistyy ja alkaa jakaa pyyntöjä työfunktioille työstettäväksi (kuva 13). Työfunktiot hyödyntävät erillisen Libuv-kirjaston säievarantoa samanaikaisten pyyntöjen suorittamiseen rinnakkain. Libuv-kirjasto on C-ohjelmointikielellä kirjoitettu kirjasto asynkronisten, eli samanaikaisten, I/O-tyyppisten (input/output) tapahtumien tukemiseen. Työfunktion suorittaessa pyynnön valmiiksi se informoi tapahtumasilmukan pääsäiettä, joka suorittaa pyyntöön rekisteröidyn callback-funktion ja palauttaa tuloksen lopulta asiakkaalle. (34; 35; 36.)



Kuva 13. Node.js-palvelimen elinkaari.

Express.js

Node.js sisältää oletuksena npm-paketinhallintasovelluksen, jota käytetään kolmannen osapuolen Node.js-ohjelmien asentamiseen ja hallinnoimiseen (35). Tämän web-sovelluksen Node.js-palvelimella käytetään Express.js-sovelluskehystä, jota hallinnoidaan npm-paketinhallintasovelluksella.

Kuten muidenkin sovelluskehysten tarkoituksena on helpottaa ohjelmointia, myös Express.js-sovelluskehys helpottaa Node.js-palvelimen toimintojen ohjelmointia. Voidaan ajatella, että web-sovelluksen Node.js-palvelin koostuu kolmesta päätasosta. Alimmalle tasolle sijoittuu Node.js:n HTTP-moduuli, keskimmäiselle tasolle sijoittuu pyynnönkäsit-

telijöitä ja ylimmälle tasolle sijoittuu reititys. Tässä yhteydessä reitityksellä tarkoitetaan sitä, miten palvelin on ohjelmoitu vastaamaan tiettyihin reitteihin, eli URL-osoitteisiin (Uniform Resource Locator), saapuviin pyyntöihin. Tämä sama rakenne suunniteltiin myös tähän web-sovellukseen käytettäväksi. (37; 38.)

```
const app = new(require('express'))();

app.use(function(req, res, next) {
   res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Con-
tent-Type, Accept, authorization");
   res.header("Access-Control-Allow-Credentials", "true");
   next();
});

function errorHandler(err, req, res, next) {
   res.status(err.actionDetails.status_code).json({ code:
err.actionDetails.status_code, error: err.actionDetails.status_details.message });
}

app.use(errorHandler);

app.use('/auth', require('./routes/auth'));
app.use('/users', require('./routes/users'));
app.use('/data', require('./routes/data'));

app.listen(3000);
```

Koodiesimerkki 3. Osa insinöörityön web-sovelluksen Node.js-palvelimen lähdekoodista.

Pyynnönkäsittelijäfunktiot ja reititys on kuitenkin helpompi tehdä Express-sovelluskehystä hyödyntäen. Koodiesimerkissä 3 näkyy osa tämän web-sovelluksen JavaScriptillä kirjoitetusta Node.js Express HTTP-palvelimen lähdekoodista. Riveillä 3–15 on määritelty kaksi pyynnönkäsittelijäfunktiota, joista ensimmäisessä määritellään sallittavat HTTP-pyyntöjen tunnisteet ja jälkimmäisessä määritellään virheenkäsittelijäfunktio. Riveillä 17–19 taas on määritelty reititys.

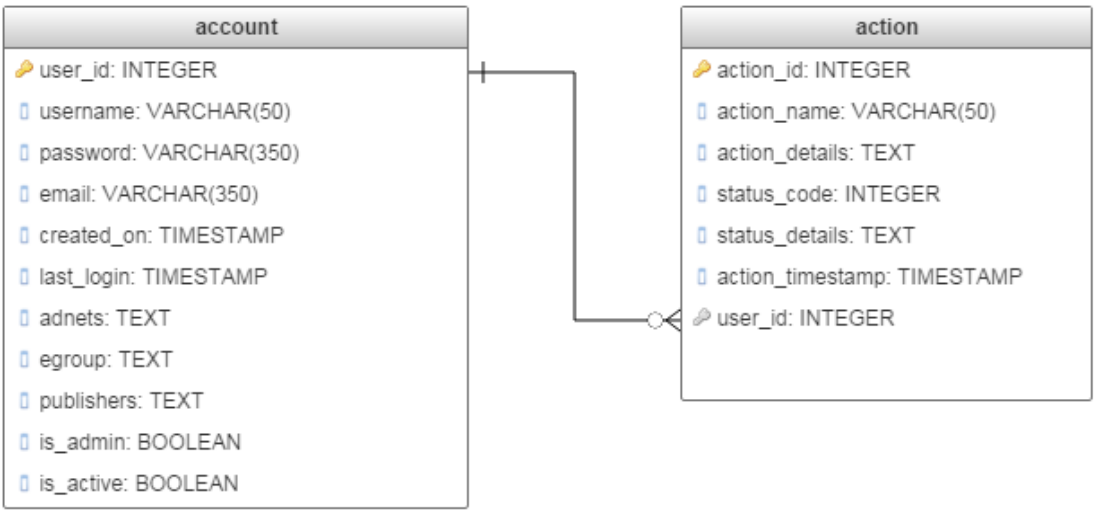
PostgreSQL

Insinöörityötä varten tarvittiin tietokanta, johon pystytään tallentamaan web-sovelluksessa luodut käyttäjät ja tehtyjen toimintojen lokitiedot. Tietokannaksi valittiin PostgreSQL. Se on avoimen lähdekoodin olio-relaatiotietokantapalvelin, jonka operointiin käytetään SQL-kyselykieltä (Structured Query Language). PostgreSQL on tunnettu kattavista ominaisuuksistaan ja luotettavuudestaan. Se kattaa muun muassa suurem-

man osan SQL-2003-standardista kuin esimerkiksi tunnetumpi MySQL-relaatiotietokanta. Amazonin pilvipalvelinympäristö tarjoaa myös helposti käyttöönotettavan PostgreSQL-palvelimen, jota juuri tässä insinöörityössäkin käytetään. (39.)

Relaatiotietokannat muodostuvat kentistä (field), jotka muodostavat rivejä eli tietueita (record), jotka taas muodostavat tauluja (table). Kentille määritellään tietotyyppi (data-type), joka määrittelee, minkä muotoista tietoa niihin voi tallentaa. Tietotyyppi voi olla esimerkiksi numeerinen, alfanumeerinen eli teksti, päivämäärä, kellonaika, totuusarvo tai bittijono. Kenttien tiedoille voidaan myös määrittää muun muassa pituus tai määritellä, onko kyseinen tieto pakollinen. Jokaiselle taululle määritellään perusavain (primary key), joka yksilöi kyseessä olevan taulun tietueet. Perusavaimena voidaan käyttää esimerkiksi juoksevia numeroita. (40.)

Relaatiotietokannan tauluista voidaan viitata toiseen tauluun perusavaimen ja viiteavaimen (foreign key) avulla. Tätä yhteyttä kutsutaan viite-eheydeksi, joka määrää, että jokaista viitattavassa taulussa olevaa viiteavaimen arvoa pitää vastata sama perusavaimen arvo kohteena olevassa taulussa. Kuvassa 14 näkyy tämän web-sovelluksen tietokantarakenne. Kuvan mukaisesti account-taulun perusavain on user_id ja vastaavasti action-taulun viiteavain on user_id. Tämän määrää, että jokaisella tehdyllä toiminnolla (action) on joku tekijä (account).



Kuva 14. Insinöörityön web-sovelluksen postgreSQL-tietokantarakenne.

4.4 Turvallisuus

Kuten kaikissa käyttäjä- ja asiakastietoja käsittelevissä web-sovelluksissa, myös insinöörityön web-sovelluksessa turvallisuus on yksi tärkeä asia, joka pitää ottaa huomioon. Koska web-sovellus vaatii sisäänkirjautumisen, on kirjautumistiedot tallennettava tietokantaan. Tietokantaan ei kuitenkaan haluta tallentaa käyttäjien salasanoja luettavassa muodossa, vaan ne tallennetaan salatussa muodossa eli käytännössä satunnaisista merkeistä koostuvissa merkkijonoissa. Tämä on varotoimenpide siltä varalta, että ulkopuolinen taho pääsisi käsiksi tietokantaan tallennettuihin tietoihin. Jos näin kuitenkin kävisi, se saisi tietoonsa vain salatut salasanat.

Bcrypt

Tässä web-sovelluksessa salasanat on salattu bcrypt-salaustekniikalla, joka perustuu Blowfish-salakirjoitukseen. Salasanojen salaus tapahtuu node.bcrypt.js-moduulin avulla Node.js-palvelimella, kun käyttäjälle luodaan salasana tai kun käyttäjä vaihtaa salasanaansa. Salasanan salaamisessa käyttäjän valitsemaan salasanaan lisätään suola, eli satunnainen merkkijono, joka vahventaa salaustekniikkaa. Ainoastaan salattu salasana tallennetaan tietokantaan. Kun käyttäjä syöttää salasanansa kirjautuessaan sisään, node.bcrypt.js-moduuli vertaa käyttäjän syöttämää salasanaa tietokantaan tallennettuun salasanaan, ja jos ne täsmäävät, käyttäjän annetaan kirjautua sisään sovellukseen. (41.)

JSON Web Token

Sisäänkirjautumisen, eli käyttäjän todentamisen (authentication), jälkeen käyttäjän suorittamat toiminnot myös varmistetaan (verify). Tätä varten valittiin käytettäväksi JSON Web Tokenit (JWT). JWT on kompakti ja omavarainen tapa välittää tietoa osapuolten välillä turvallisesti JSON-muodossa. JWT:n sisältämä tieto voidaan todentaa, koska se on digitaalisesti allekirjoitettu. Tässä web-sovelluksessa JWT allekirjoitetaan salaisella avaimella. Kun käyttäjä kirjautuu onnistuneesti sisään sovellukseen, hänen selaimen paikalliseen tallennustilaan (local storage) tallennetaan JWT 24 tunnin ajaksi. Käyttäjä pystyy käyttämään sovelluksen toimintoja niin kauan, kuin JWT on voimassa, minkä jälkeen hänen on kirjauduttava uudestaan sisään. (42.)

HTTPS-protokolla

Edellä lueteltujen turvallisuustoimenpiteiden lisäksi käyttäjän verkkoselaimen ja web-sovelluksen palvelimen välinen tiedonsiirto on syytä suojata. HTTPS-protokolla (Hypertext Transfer Protocol Secure) on muutoin sama kuin HTTP-protokolla, mutta se tunneloidaan turvallisen SSL/TLS-salausprotokollan läpi. Tämä suojaa tiedonsiirtoa käyttäjän verkkoselaimen ja sovelluksen palvelimen välillä. Tässä insinöörityössä käytetään HTTP-protokollaa kehitysvaiheessa, mutta ennen tuotantovaiheeseen menoa tiedonsiirto on suunniteltu siirrettäväksi tapahtumaan HTTPS-protokollan välityksellä. (43.)

SQL-injektio

Yksi yleisimmistä web-ympäristön hakkerointitekniikoista on SQL-injektio, jolla voidaan tuhota tietokantoja. Koska tässä web-sovelluksessa käytetään tietokantaa, tulee sovellus suojata myös SQL-injektioilta. SQL-injektio tarkoittaa sitä, että hyökkääjä onnistuu syöttämään sovelluksen tietokantaan SQL-komentoja, joita ei pitäisi pystyä syöttämään. Tämä voi tapahtua esimerkiksi sovelluksessa olevan suojaamattoman lomakkeen kautta, johon pyydetään syöttämään esimerkiksi käyttäjätunnus, mutta hyökkääjä syöttääkin SQL-komentoja, jotka päätyvät tietokantaan asti. (44.)

Tässä web-sovelluksessa käytetään schema-inspector-moduulia, jota hyödyntäen kaikkiin sovelluksen lomakkeisiin syötetyt tiedot siistitään ja tarkistetaan palvelinpuolella ennen tietokantaan syöttämistä. (45.)

5 Web-sovelluksen toteutus

Web-sovelluksen suunnittelun jälkeen lähdettiin miettimään, miten sovellus olisi järkevintä toteuttaa. Kehitystyö päätettiin aloittaa kehitysympäristön rakentamisesta. Seuraavaksi luotaisiin tietokanta, jotta käyttäjätietoja voidaan tallentaa ja näin myös palvelinpuolen API-päätepisteet olisi mahdollista suojata. Lopuksi web-sovellus rakennettaisiin pala kerrallaan palvelin- ja selainpuolen kehitystä vuorotellen. Ensin palvelinpuolelle luodaan API-päätepiste, ja tämän jälkeen selainpuolelle käyttöliittymään mahdollisuus kutsua kyseistä päätepistettä ja esittää sen palauttama data.

5.1 Kehitysympäristö

Kehitysympäristön rakentaminen alkoi Node.js-suoritusympäristön asentamisesta, Webpackin konfiguroinnista ja tarvittavien moduulien asentamisesta npm-paketinhallintasovellusta ja komentoriviä käyttäen. Esimerkiksi Webpackin asentaminen kehitysympäristöön onnistuu yksinkertaisesti komentorivin komennolla ”npm install webpack –save”, kunhan Node.js ja npm ovat ensin asennettuina.

Palvelin- ja selainpuoli päätettiin jakaa omiin kansiorakenteisiinsa sovelluksen rakenteen selkeyttämiseksi. Webpack sisältää kehityspalvelimen, jota hyödyntämällä sovelluksen selainpuoli saadaan käyntiin paikallisesti localhost-osoitteeseen, joten erillistä web-palvelinta ei tarvitse käyttää. Selainpuolen kehitystyön helpottamiseksi Webpack konfiguroitiin käyttämään react-hot-loader-moduulia, joka tarkkailee selainpuolen lähdekoodin muutoksia ja päivittää uudet muutokset tallentamishetkellä automaattisesti selaimeen ilman erillistä sivulatausta. Tämän ansiosta muun muassa web-sovelluksen senhetkinen tila ei katoa, jos selainpuolen lähdekoodiin tehdään muutoksia.

Vastaavasti palvelinympäristö käynnistetään Node.js-suoritusympäristöön nodemon-moduulin välityksellä. Nodemon-moduuli tarkkailee muutoksia palvelinpuolen lähdekoodissa ja uudelleenkäynnistää palvelimen automaattisesti havaitessaan muutoksen. Käytännössä siis kun lähdekoodiin tehdään muutos ja se tallennetaan, nodemon käynnistää palvelimen uudestaan, jolloin uudet muutokset astuvat voimaan.

Jotta selain- ja palvelinpuoli saatiin käynnistettyä helposti, määriteltiin projektiin skripti, joka käynnistää molemmat samanaikaisesti. Skripti määriteltiin siten, että kun komentoriville kirjoitettaan ”npm run start”, hetken kuluttua sekä selain- että palvelinpuolen projektit ovat paikallisesti käynnissä ja valmiina ottamaan muutoksia vastaan.

Web-sovelluksen lähdekoodin versionhallintaan käytettiin Git-pohjaista Bitbucket-versionhallintaohjelmaa (https://bitbucket.org/product) (46). Versionhallintaohjelmistolla sovelluksen kaikki kehitysvaiheet ja koodimuutokset pysyvät tallessa. Jos esimerkiksi haluttaisiin kumota jokin muutos tai palata tiettyyn pisteeseen takaisin, se onnistuu helposti palaamalla versiohistoriassa taaksepäin. Bitbucket valittiin käytettäväksi, koska se tarjoaa ilmaiseksi yksityisiä lähdekoodin säilytyspaikkoja, jotka eivät ole muiden tarkasteltavissa. Tämän web-sovelluksen sekä selain- että palvelinpuolen lähdekoodi tallennettiin Bitbucketiin lukuun ottamatta API- ja muita salaisia avaimia. Sen lisäksi, että lähdekoodi pysyy tallessa versionhallintaohjelmassa, se mahdollistaa esimerkiksi kehi-

tystyön tekemisen tarvittaessa usealta eri tietokoneelta, koska lähdekoodi on aina saatavilla.

5.2 Tietokanta

Tämän web-sovelluksen ainoa suojaamaton API-päätepiste on kirjautumistoimintoon vaadittava polku. Ennen sen toteuttamista tarvittiin käyttäjätiedoille tietokanta käyttäjätunnuksen ja salasanan tarkistamiseksi. Käyttäjätiedoissa määriteltiin myös, mihin mainontahallintajärjestelmän tietoihin käyttäjällä on pääsy. Amazonin pilvipalvelinympäristöön luotiin postgreSQL-instanssi, johon tietokanta rakennettiin.

Kuvassa 14, sivulla 22, on web-sovelluksen postgreSQL-tietokannan rakenne kuvattuna. Tietokanta koostuu kahdesta taulusta, jotka ovat account ja action. Account-tauluun tallennetaan kunkin käyttäjätunnuksen tiedot, joita ovat muun muassa käyttäjätunnus, salasana ja sähköpostiosoite. Tauluun haluttiin myös tallentaa aikaleimat käyttäjätunnuksen luomishetkestä ja viimeisestä sisäänkirjauksesta. Oleellinen osa käyttäjätietoja ovat myös mainonnanhallintajärjestelmään liittyvät oikeudet, jotka tallennetaan riveille adnets, egroup ja publishers. Lisäksi tauluun lisättiin vielä totuusarvon sisältämät rivit is_admin ja is_active, jotka ilmaisevat, onko kyseessä ylläpitäjä ja onko tunnus aktiivinen.

Kuvan 14 toiseen tauluun (action) tallennetaan käyttäjien tekemät toiminnot websovelluksessa. Näitä toimintoja ovat esimerkiksi sisäänkirjautuminen ja mainospaikkojen hakeminen. Action-tauluun tallennetaan jokaisen toiminnon yhteydessä yksityiskohtia toiminnosta, joita ovat esimerkiksi haetut mainospaikat. Lisäksi haluttiin tallentaa toiminnon aikaleima ja tila, eli se, onnistuiko toiminnon suorittaminen vai ei.

Tietokannan taulujen viite-eheys muodostuu account-taulun user_id-perusavaimen ja action-taulun user_id-viiteavaimen välille. Näin ollen jokaiselle suoritetulle toiminnolle löytyy aina toiminnon suorittama käyttäjätunnus tietokannasta.

Tietokantaan syötettiin aluksi manuaalisesti yhden admin-käyttäjän tiedot, jotta kehitysvaihe saatiin käyntiin. Jälkeenpäin oli tarkoitus luoda palvelinpuolelle API-päätepiste käyttäjätietojen lisäämiseksi ja muokkaamiseksi. Käyttäjätunnuksia ei haluta missään vaiheessa poistaa kokonaan historiatietojen säilyttämiseksi. Tämän takia jokainen käyt-

täjätunnus voidaan määritellä aktiiviseksi tai inaktiiviseksi. Jos tunnus halutaan pois käytöstä, se muutetaan inaktiiviseksi.

5.3 Selain- ja palvelinpuolen kehitys

Tarkoituksena oli ensiksi määritellä palvelinpuolelle Express-sovelluskehyksen avulla API-päätepisteitä, joita selainpuoli kutsuu toimintojen suorittamiseksi. Selain- ja palvelinpuolen kehitystä vuoroteltiin siten, että ensin palvelinpuolelle luotiin API-päätepiste ja tämän jälkeen selainpuolelle käyttöliittymään mahdollisuus hyödyntää päätepistettä ja esittää sen palauttama data.

Kirjautumistoiminto

Ensimmäiseksi palvelinpuolelle määriteltiin API-päätepiste polkuun /auth/getToken/, jota kutsumalla suoritetaan sisäänkirjautuminen. Tämä API-päätepiste suunniteltiin siten, että selain lähettää HTTP POST -pyynnön palvelimelle, johon on liitetty käyttäjätunnus ja salasana. Palvelin tarkistaa, löytyykö tietokannasta kyseistä käyttäjätunnusta ja täsmääkö salasana käyttäjätunnukselle. Jos käyttäjätunnus ja salasana ovat oikein, palvelin luo JWT:n, johon kirjataan kyseinen käyttäjätunnus ja voimassaoloaika, ja palauttaa sen selaimelle. Selain tallentaa JWT:n paikalliseen muistiin ja siirtyy sisälle käyttöliittymään. Jos kirjautumistiedot ovat väärin, palvelin lähettää virheviestin selaimelle käyttäjän nähtäville. Käyttäjän kirjautuessa ulos sovelluksesta käyttäjän selaimesta poistetaan JWT ja käyttäjä ohjataan takaisin sisäänkirjautumissivulle.

Mainospaikkojen hakeminen

Mainospaikkojen hakemista varten palvelinpuolelle määriteltiin API-päätepiste polkuun /data/getSiteNetworks/. Selaimen kutsuessa HTTP GET -pyynnöllä päätepistettä, palvelin palauttaa kaikki kyseiselle käyttäjälle sallitut mainospaikat Cxense Display -mainonnanhallintajärjestelmästä. HTTP GET -pyyntöön liitetään authorization-tunniste, jonka arvoksi määritellään käyttäjän JWT. Ilman voimassa olevaa JWT:tä palvelin palauttaa pelkän virheviestin, mikä estää päätepisteen käyttämisen ilman onnistunutta sisäänkirjautumista. Tätä samaa logiikkaa käytetään myös web-sovelluksen muissa suojatuissa API-päätepisteissä.

Jotta palvelimen olisi mahdollista palauttaa mainospaikkatietoja selaimelle, palvelimen täytyy tehdä kysely Cxense Displayn rajapintaan halutuilla parametreilla ja palauttaa tulos lopulta JSON-muodossa takaisin. Cxense Display on sen verran vanha järjestelmä, että sen rajapinta palauttaa tuloksia ainoastaan teksti- tai XML-muodoissa. Palvelimen tuli siis muuntaa XML-muotoinen tulos JSON-muotoon ennen sen palauttamista selaimelle. Muuntamisessa hyödynnettiin valmista node-xml2js-moduulia (47), joka tarjosi valmiin funktion muuntamiselle. Dataa haluttiin käsitellä JSON-muodossa, koska sen esittäminen React-pohjaisessa käyttöliittymässä on helppoa.

Cxense Displayssä oleva tieto on hierarkiamuodossa siten, että se koostuu julkaisijoista, joiden alla on osioita, joiden alla taas on mainospaikkoja. Cxense Displayn rajapinta ei tarjoa valmiina toimintoa näiden kaikkien tietojen hakemiselle yhdellä kerralla, mikä hieman yllättäen monimutkaisti tämän toiminnallisuuden tekemistä. Koska web-sovelluksen käyttäjätiedot määriteltiin siten, että jokaiselle käyttäjälle määritellään keiden julkaisijoiden tietoihin käyttäjällä on pääsy, tuli Cxense Displayn rajapintaa loppujen lopuksi kutsua kolme kertaa, jotta kaikki tarvittavat tiedot saatiin haettua selainpuolta varten.

Ensin Cxense Displaystä haetaan julkaisijatiedot, minkä tuloksen pohjalta voidaan hakea osiotiedot, minkä pohjalta voidaan lopulta hakea mainospaikkatiedot. Nämä tiedot haluttiin vielä lopuksi yhdistää yhdeksi JSON-objektiksi, jotta niiden esittäminen olisi mahdollisimman yksinkertaista selaimessa. Selaimen oli tarkoitus näyttää mainospaikat loogisesti oikeiden osioiden ja julkaisijoiden alla listattuna, jotta käyttäjä tietää, mitä on tarkastelemassa.

Koodiesimerkissä 4 näkyy osa toiminnoista, joita palvelimella suoritetaan, kun selain tekee HTTP GET -pyynnön palvelimen /data/getSiteNetworks/-polkuun. Riveillä 2–10 varmistetaan GET-pyynnön authorization-tunnisteessa oleva JWT. Jos JWT ei ole pätevä, toimintoja ei suoriteta enempää ja selaimelle palautetaan virheilmoitus. Riveillä 11–22 web-sovelluksen tietokantaan tehdään kysely, jossa käyttäjätunnuksen tiedoista haetaan julkaisijatiedot, jotka määrittelevät, mihin Cxense Displayn julkaisijatietoihin käyttäjällä on pääsy. Riveillä 23–28 Cxense Displaysta haetaan käyttäjän oikeuksien mukaisesti julkaisijat, joiden perusteella haetaan vastaavat osiot, joiden perusteella haetaan lopuksi vastaavat mainospaikat. Riveillä 29–33 Cxense Displayn palauttama XML-muotoinen data muutetaan JSON-muotoiseksi. Riveillä 34–41 JSON-muotoinen data yhdistetään, ja rivillä 42 JSON-muotoinen data palautetaan selaimelle. Riveillä

43–46 on yksi virheenkäsittelyfunktiosta, joka virheen sattuessa palauttaa virheilmoituksen selaimelle.

```
router.get('/getSiteNetworks', (req, res, next) => {
    let token = req.headers['authorization'];
    try {
        let decoded = api.auth.verifyToken(token);
        user.username = decoded.username;
        actionDetails.username = decoded.username;
    } catch (err) {
        actionDetails.status_code = 401;
        return next({ actionDetails: actionDetails });
    }
    let queryObj = account.select(account.user_id, account.egroup, ac-
count.publishers).from(account).where(account.username.equals(user.username
)).toQuery();
    db.query(queryObj.text, queryObj.values)
        .then(function(data) {
            if (data.length === 0) {
                actionDetails.status_code = 500;
                return next({ actionDetails: actionDetails });
            } else {
                actionDetails.user_id = data[0].user_id;
                return { egroup: data[0].egroup, publishers: data[0].publishers };
            }
        }).then(function getSiteNetworkData(data) {
            funcs.push(cxense.retrieveDataAsync('publisher', data.egroup, da-
ta.publishers));
            funcs.push(cxense.searchDataAsync('section', data.egroup, null));
            funcs.push(cxense.searchDataAsync('cu', data.egroup, null));
            return Promise.all(funcs);
        }).then(function xmlsToJsons(xmls) {
            for (let xml of xmls) {
                funcs.push(cxense.xmlToJsonAsync(xml));
            }
            return Promise.all(funcs);
        }).then(function handleEASResponseAndSendData(responses) {
            for (let response of responses) {
                responseData.push(api.utils.handleEASresponse(response));
            }
            let sections = api.utils.combineSectionsAndCUS(responseData[1], re-
sponseData[2]);
            let siteNetworks =
api.utils.combinePublishersAndSections(responseData[0], sections);
            return res.status(200).json({ data: siteNetworks });
        }).catch(function(error) {
            actionDetails.status_code = 500;
            return next({ actionDetails: actionDetails });
        });
});
```

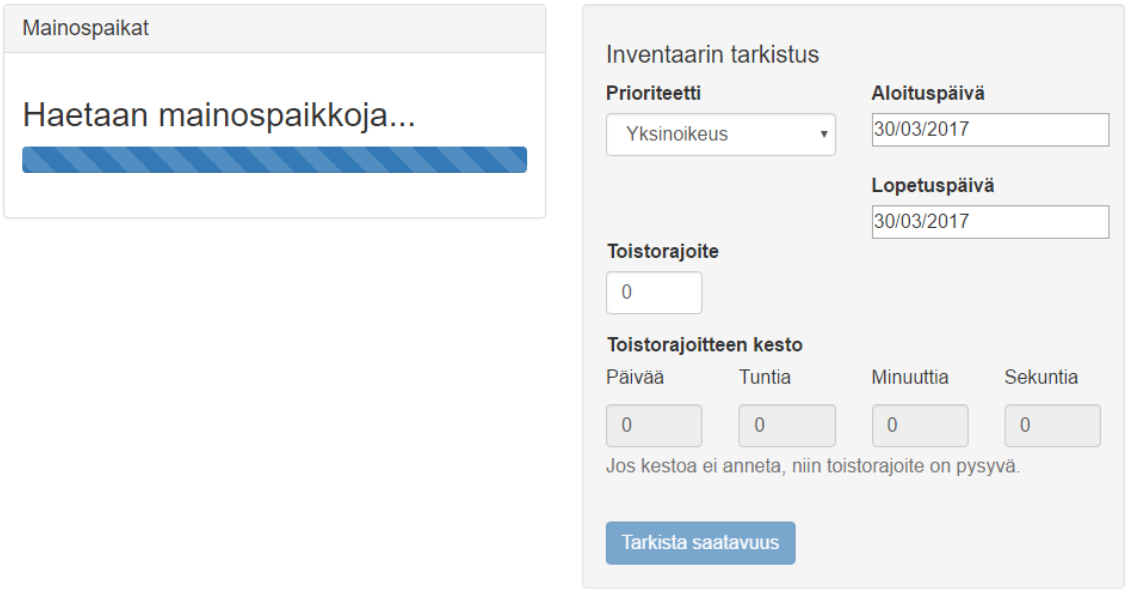
Koodiesimerkki 4. Palvelinpuolen /data/getSiteNetworks/-polussa suoritettavat toiminnot.

Koko tapahtumaketju mainospaikkojen hakemiseen lähtee liikkeelle automaattisesti, kun käyttäjä kirjautuu sisään web-sovellukseen ja siirtyy mainosinventaarinäkymään. Mainosinventaarinäkymässä suoritetaan getSiteNetworks-funktio (koodiesimerkki 5), joka lähettää HTTP GET -pyynnön palvelimen /data/getSiteNetworks/-polkuun ja useita toimintoja selainpuolella käytössä olevan Redux-kirjaston Reducerille. Koodiesimerkin 5 rivillä 4 lähetetään ensimmäinen fetchSiteNetworksRequest-toiminto Reducerille. Tällöin Reducer palauttaa sovellukselle uuden tilan, jossa on määritelty, että mainospaikkojen haku on käynnissä. Tämän perusteella taas React-komponentti päivittyy siten, että käyttäjälle näytetään latauspalkki mainospaikkalistauksen kohdalla käyttöliittymässä (kuva 15).

```
function getSiteNetworks(token) {
   return (dispatch, state) => {
      let fetchUrl = baseUrl + '/data/getSiteNetworks';
      dispatch(fetchSiteNetworksRequest());
      return fetch(fetchUrl, {
            credentials: 'include',
            headers: {
               'Authorization': `Bearer ${token}`
            }
         })
         .then(checkHttpStatus)
         .then(parseJSON)
         .then(response => {
            dispatch(receiveSiteNetworks(response.data));
         })
         .catch(error => {
            if (error.code === 401) {
               dispatch(authActionCreators.loginUserFailure({ status: error.code,
statusText: error.message }));
               dispatch(push('/login'));
            } else {
               dispatch(fetchSiteNetworksError(error));
            }
         })
   }
}
```

Koodiesimerkki 5. Selainpuolen funktio mainospaikkojen hakemiselle.

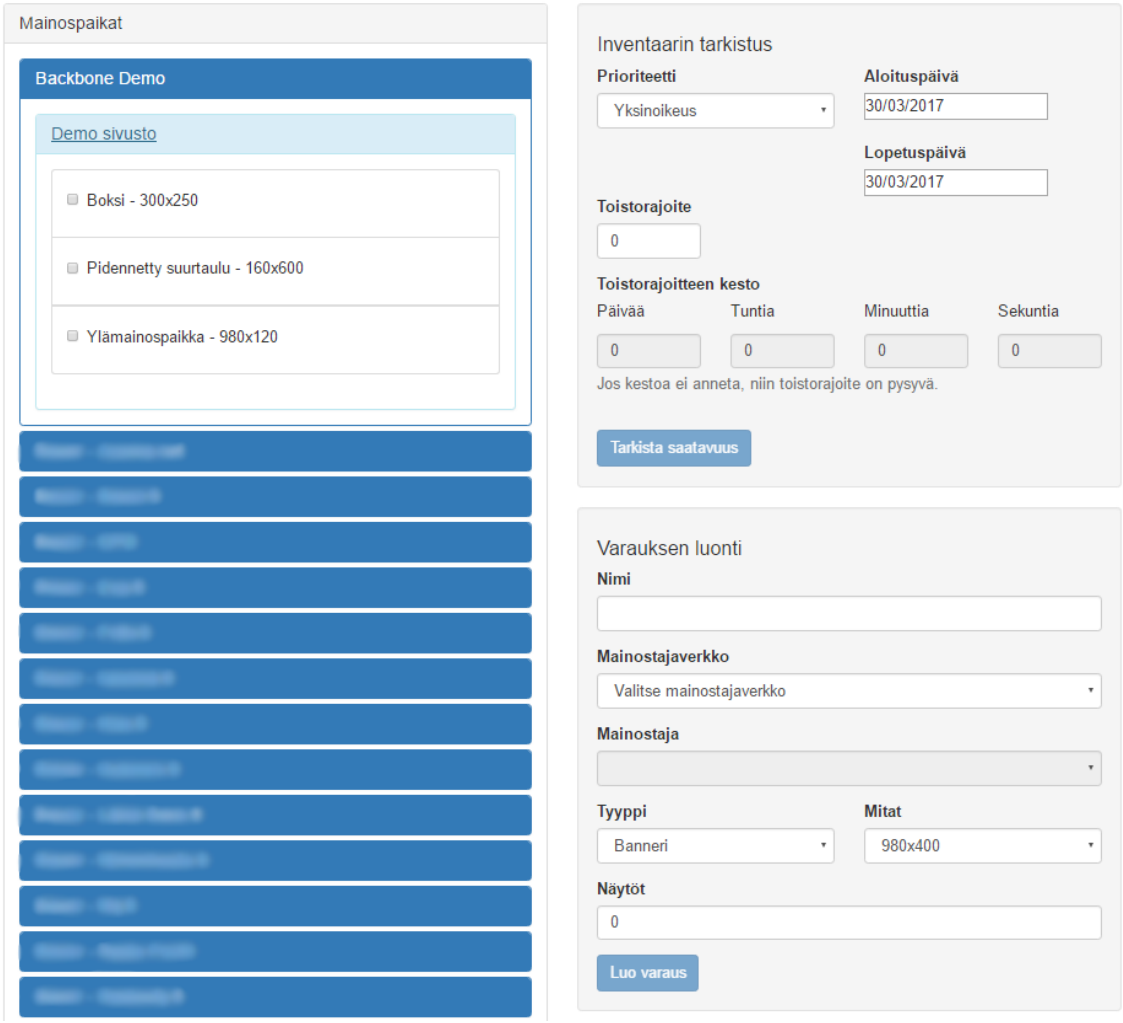
Koodiesimerkin 5 riveillä 5–10 getSiteNetworks-funktio lähettää HTTP GET -pyynnön palvelimelle fetch-funktiota hyödyntäen. Rivillä 8 määritellään authorization-tunnisteeseen käyttäjän JWT. Tämän jälkeen selain odottaa, että palvelin suorittaa koodiesimerkissä 4 nähtävät toiminnot. Kun palvelin palauttaa tuloksen selaimelle, koodiesimerkin 5 getSiteNetworks-funktiossa siirrytään riville 11, jossa tarkistetaan palvelimen palauttama HTTP-tila.



Kuva 15. Inventaarinäkymä, kun mainospaikkoja haetaan.

Jos HTTP-tila on 200, eli OK, funktiossa siirrytään riville 12, jossa jäsennetään palvelimelta saatu JSON-muotoinen data. Lopuksi siirrytään riveille 13–15, jossa Reducerille lähetetään receiveSiteNetworks-toiminto jäsennetyn datan kanssa, minkä jälkeen Reducer palauttaa sovellukselle taas uuden tilan, jossa on määritelty haetut mainospaikat ja se, että mainospaikkojen hakeminen on päättynyt. Tämän jälkeen React-komponentti päivittyy siten, että mainospaikkalistauksen kohdalla olleen latauspalkin tilalle listataan mainospaikat (kuva 16).

Jos esimerkiksi käyttäjän JWT olisi vanhentunut tai palvelinpuolella olisi tapahtunut jokin virhe, palvelin olisi palauttanut virhekoodin, jolloin getSiteNetworks-funktiossa olisi siirrytty suoraan virheenkäsittelyriveille 16–23. Riippuen tapahtuneesta virheestä käyttäjää olisi joko pyydetty kirjautumaan sovellukseen uudestaan sisään tai hänelle olisi näytetty lyhyt viesti tapahtuneesta virheestä.



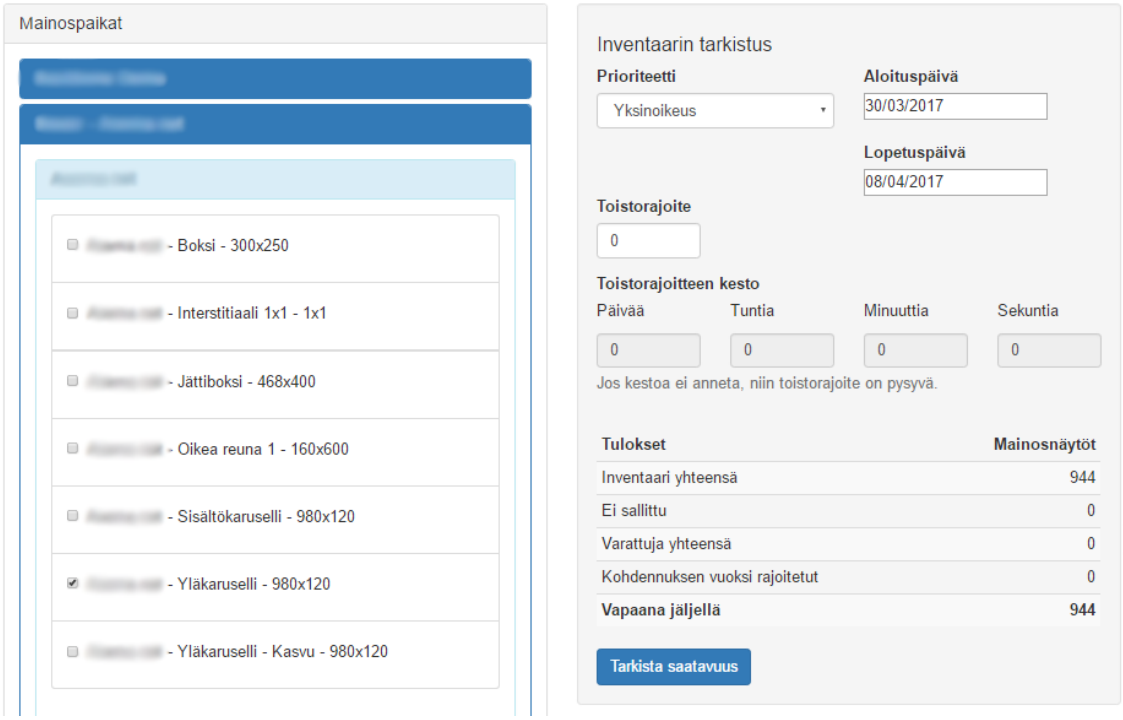
Kuva 16. Inventaarinäkymä, kun mainospaikat on haettu.

Mainosinventaarin tarkistus ja mainoskampanjan varaus

Mainosinventaarin tarkistuksessa ja mainoskampanjan varaamisessa käytettiin hyvin samankaltaista logiikka kuin mainospaikkojen hakemisessakin. Ensin palvelinpuolelle määriteltiin suojatut API-päätepisteet polkuihin /data/getAvailability/ (mainosinventaarin tarkistus) ja /data/addCampaign/ (mainoskampanjan varaaminen) ja niissä suoritettavat palvelinpuolen toiminnot. Tämän jälkeen selainpuolen käyttöliittymään luotiin vastaavasti osiot näiden toimintojen suorittamiseen.

Kuvassa 16 näkyy Inventaarin tarkistus -osio, jossa käyttäjä voi valita prioriteetin, ajanjakson ja toistorajoitteen. Lisäksi käyttäjän tulee valita vasemmalla olevasta Mainospaikat-osiosta tarkistettavat mainospaikat. Kun asetukset on valittu, Tarkista saatavuus -painike aktivoituu ja käyttäjä voi klikata sitä.

Mainosinventaarin tarkistuksen tapahtumaketju lähtee liikkeelle käyttäjän klikatessa painiketta. Tällöin selain lähettää HTTP GET -pyynnön palvelimelle, jossa on mukana valitut asetukset ja käyttäjän JWT. Ensin palvelin varmistaa JWT:n pätevyyden, minkä jälkeen se hakee Cxense Displaysta tarvittavan tiedon, jonka se lopulta palauttaa takaisin selaimelle ja käyttäjälle näytettäväksi (kuva 17).



Kuva 17. Inventaarinäkymä, kun mainosinventaarin tarkistus on suoritettu.

Kuvan 16 Varauksen luonti -osiossa käyttäjä suorittaa mainoskampanjan varaamisen, kun hän on ensin tarkistanut mainosinventaarin. Varauksen luontia varten käyttäjän tulee syöttää käyttöliittymään mainoskampanjan nimi ja valita mainostaja, kampanjatyyppi ja kampanjan mitat. Näytöt-kenttään asetetaan arvo automaattisesti, kun mainosinventaarin tarkistus on suoritettu, mutta käyttäjä voi halutessaan muokata sitä.

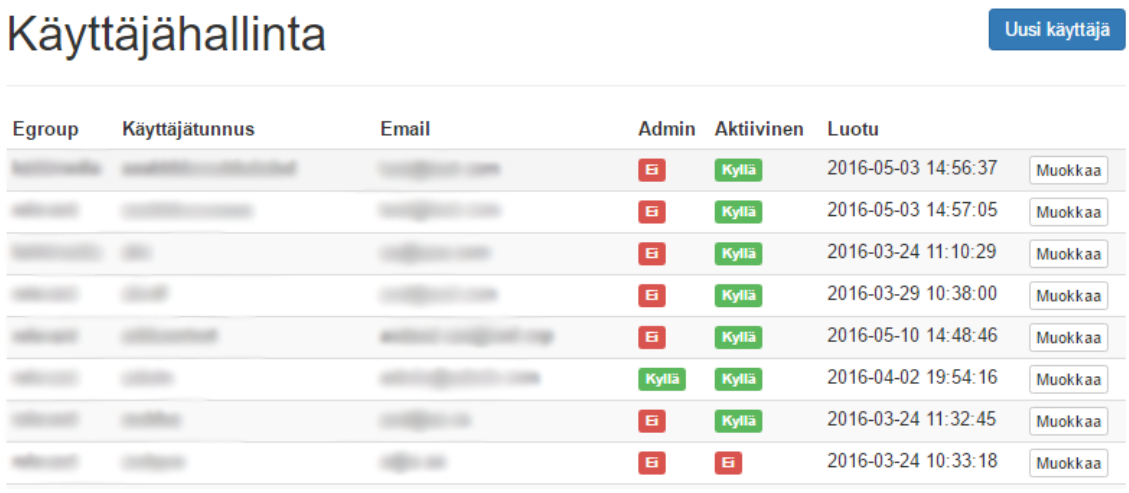
Käyttäjän klikatessa Luo varaus -painiketta selain lähettää HTTP POST -pyynnön palvelimelle, jossa on mukana valitut mainoskampanja-asetukset ja käyttäjän JWT. Myös tässä tapauksessa palvelin ensin tarkistaa JWT:n pätevyyden, minkä jälkeen palvelin lähettää Cxense Displayyn tiedot luotavasta mainoskampanjasta. Lopuksi palvelin lähettää selaimelle tiedon varauksen onnistumisesta tai epäonnistumisesta, minkä jälkeen käyttäjää informoidaan tuloksesta. Jos mainoskampanjan luonti onnistui, mainosinventaari on varattu Cxense Displayssa.

Käyttäjähallinnan toiminnot

Lopuksi luotiin vielä toiminnallisuudet salasanan vaihtamiselle, käyttäjän lisäämiselle ja käyttäjän muokkaamiselle. Jokaisen käyttäjän oli tarkoitus pystyä vaihtamaan oma salasana, mutta käyttäjän lisäys ja muokkaus oli jo suunnitteluvaiheessa rajattu vain ylläpitäjien toiminnoiksi. Näitä toimintoja varten palvelinpuolelle luotiin polut /users/changePassword/ (salasanan vaihto), /users/addUser/ (käyttäjän lisäys), /users/editUser/ (käyttäjän muokkaus) ja /users/getUsers/ (käyttäjien haku).

Selainpuolella salasana vaihdetaan yksinkertaisen lomakkeen avulla, johon syötetään uusi salasana kahteen kertaa ja klikataan Vaihda salasana -painiketta. Painiketta klikattaessa selain lähettää HTTP POST -pyynnön palvelimelle, jossa on mukana uusi salasana ja käyttäjän JWT. Jos JWT on pätevä, palvelin salaa salasanan Bcrypt-salaustekniikalla ja syöttää uuden salatun salasanan tietokantaan vanhan tilalle. Lopuksi palvelin lähettää tulokset takaisin selaimelle ja selain ilmoittaa salasanan vaihdosta käyttäjälle.

Selainpuolen käyttöliittymän ylänavigaatiopalkkiin lisättiin painikkeita siten, että jos kirjautuneena on tavallinen käyttäjä, hän näkee ainoastaan Salasanan vaihto -painikkeen. Ylläpitäjät näkevät tämän lisäksi Käyttäjähallinta-painikkeen, jota klikkaamalla he pystyvät siirtymään käyttäjähallintanäkymään (kuva 18).

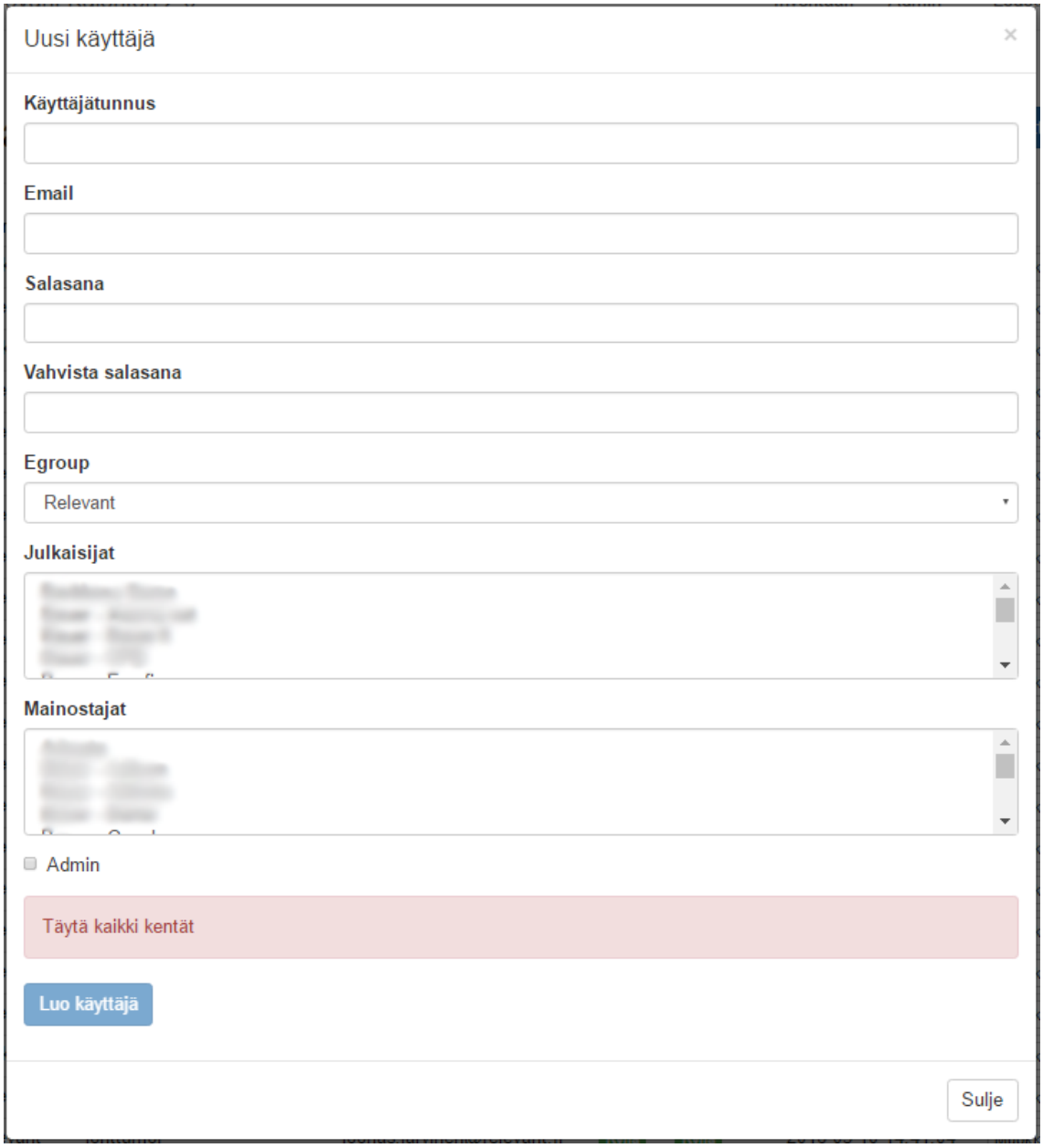


Kuva 18. Ylläpitäjän käyttäjähallintanäkymä.

Ylläpitäjän siirtyessä käyttäjähallintanäkymään ensimmäistä kertaa sisäänkirjautumisen jälkeen selain lähettää automaattisesti HTTP GET -pyynnön palvelimen

/users/getUsers/-polkuun. Palvelin tarkistaa ensiksi, että ylläpitäjän JWT on pätevä, minkä jälkeen palvelin tarkistaa vielä tietokannasta, että käyttäjä on oikeasti ylläpitäjä. Vasta tämän jälkeen palvelin hakee tietokannasta kaikki sovelluksen käyttäjät ja palauttaa ne selaimelle ja lopulta ylläpitäjälle nähtäväksi.

Käyttäjähallintanäkymä luotiin niin, että ylläpitäjän on mahdollista lisätä uusia käyttäjiä ja muokata vanhoja käyttäjiä samasta näkymästä. Sekä Uusi käyttäjä- että Muokkaa-painikkeet avaavat näkymän päälle modaalin, jossa toiminnot voi suorittaa loppuun (kuva 19).



Kuva 19. Uuden käyttäjän lisäys -modaali.

Sekä uuden käyttäjän lisäämisessä että käyttäjän muokkaamisessa selain lähettää palvelimelle HTTP POST -pyynnön, jossa on mukana syötetyt asetukset ja käyttäjän JWT. Kuten käyttäjien hakemisessa, myös näissä toiminnoissa palvelin tarkistaa ensin JWT:n pätevyyden ja ylläpitäjän oikeuden tietokannasta ennen varsinaisten käyttäjämuutosten viemistä tietokantaan. Jos toiminto onnistui, käyttäjälle ilmoitetaan asiasta ja modaali suljetaan. Virhetilanteessa modaalissa näytetään tapahtunut virhe, joka voi olla esimerkiksi se, että käyttäjätunnus on jo käytössä uutta käyttäjää luotaessa.

6 Yhteenveto

Insinöörityössä suunniteltiin ja toteutettiin mainonnanhallintaa helpottavan web-sovelluksen prototyyppi Relevant Partner 4 Media Oy:lle. Web-sovelluksen käyttötarkoituksena oli helpottaa mainonnanhallintaprosessiin kuuluvia mainosinventaarin tarkistus- ja mainoskampanjan varaus -työvaiheita. Käyttötarkoituksen tuli kuitenkin olla myös tarvittaessa laajennettavissa muihin mainonnanhallintaprosessin työvaiheisiin tulevaisuutta ajatellen. Lisäksi insinöörityössä perehdyttiin web-sovelluksessa käytettäviin tekniikoihin.

Insinöörityössä toteutettu web-sovellus koostuu kolmesta eri osasta: React-pohjaisesta selainpuolesta, Node.js-pohjaisesta palvelinpuolesta ja postgreSQL-tietokannasta. Lisäksi web-sovellus keskustelee ohjelmointirajapinnan kautta ulkoisen Cxense Display -mainonnanhallintajärjestelmän kanssa.

Web-sovellus suunniteltiin helppokäyttöiseksi ja responsiiviseksi. Suunnitteluvaiheessa tehdyt rautalankamallit web-sovelluksen käyttöliittymästä helpottivat huomattavasti selainpuolen toteuttamista. Suurimmat toteutusvaiheen haasteet tulivatkin vastaan palvelinpuolella. Suunnitteluvaiheessa olisi voitu selvittää Cxense Displayn ohjelmointirajapintamahdollisuudet hieman tarkemmin muun muassa mainospaikkojen hakemisen kannalta. Ohjelmointirajapinnan tuomat haasteet saatiin ratkottua, mutta ne lisäsivät toiminnallisuuksien monimutkaisuutta. Osa toteutusvaiheen haasteista johtui myös puhtaasti kokemattomuudesta palvelinpuolen tekniikoissa, mutta niihin saatiin apua kokeneemmalta ohjelmoijalta, joka myös kommentoi kirjoitettua koodia.

Valmista prototyyppiä testattiin paikallisesti kehitysympäristössä muutaman hengen voimin eri selaimilla ja päätelaitteilla. React-Bootstrap-sovelluskehyksen tuoma responsiivisuus mahdollisti sovelluksen käytön erikokoisilla näytöillä, kuten myös älypuhe-

limilla. Varsinaisia puutteita toiminnallisuuksista ei löytynyt, mutta käyttöliittymän ulkoasu ei ollut testausvaiheessa viimeistelty, vaan käytössä oli valmiiden käyttöliittymäkomponenttien oletusulkoasut. Kovin laajamittaista testausta ei tässä vaiheessa kuitenkaan ole suoritettu.

Insinöörityön lopputulokseksi saatiin kehitysympäristöön luotu prototyyppi web-sovelluksesta. Alkuperäisestä suunnitelmasta poiketen web-sovellusta ei ole tämän insinöörityön aikana viety tuotantoympäristöön, ja näin ollen myöskään web-sovelluksen HTTP-liikennettä ei ole siirretty käyttämään HTTPS-protokollaa. Web-sovelluksen käyttöliittymän ulkoasu ei myöskään ole Relevantin yritysilmeen mukainen, vaan käytetyn React-Bootstrap-sovelluskehyksen oletusteeman mukainen. Tämä johtuu osittain siitä, että web-sovelluksen suunnitteluhetkellä Relevantin yritysilmeen uudistaminen oli meneillään, joten osuus jätettiin vielä siinä vaiheessa auki. Toiminnallisuuksiltaan web-sovellus on kuitenkin suunnitelman mukainen, joten lopputulos on vähintäänkin tyydyttävä.

Web-sovellukseen jäi tulevaisuutta ajatellen paljon kehitettävää, isoimpana tuotantovaiheeseen vienti. Web-sovelluksen tulevaisuus on kuitenkin tällä hetkellä epävarma Relevantin sisäisten muutosten vuoksi. Luultavasti sovellusta voidaan hyödyntää yrityksen sisäisesti, tai mahdollisesti sovelluksesta voidaan irrottaa komponentteja toisiin Relevantin projekteihin. Tarvittaessa sovelluksen käyttötarkoitus on myös helposti laajennettavissa esimerkiksi lisäämällä API-päätepisteitä palvelinpuolelle.

Insinöörityön ehdottomasti suurimmaksi hyödyksi muodostui ohjelmointitaitojen kehittyminen ja laaja tietämyksen lisääntyminen eri tekniikoista. Web-sovelluksen kehitysvaiheen haasteet pakottivat eri tekniikoiden soveltamisen kunkin työvaiheen mukaisesti, mikä avarsi ajattelutapaa ja ymmärrystä tekniikoista. Relevantilla on käynnissä muun muassa erilaisten käyttöliittymien kehitysprojekteja, joita ajatellen insinöörityö on tuonut hyvää kokemusta ja tietotaitoa.

Lähteet

1 Digimainonnan kasvuvauhti tuplaantui edellisvuodesta. Verkkodokumentti. IAB Finland. <http://www.iab.fi/ajankohtaista/digimarkkinoinnin-uutiset/kvartaalitiedotteet/digimainonnan-kasvuvauhti-tuplaantui-edellisvuodesta.html>. Luettu 3.2.2017.

2 Suositukset. Verkkodokumentti. IAB Finland. <http://www.iab.fi/digimainonnan-abc/suositukset-2.html>. Luettu 3.2.2017.

3 IAB Finland. Verkkodokumentti. IAB Finland. <http://www.iab.fi/iab-finland.html>. Luettu 3.2.2017.

4 Advertising Campaigns - Meaning and its Process. Verkkodokumentti. MSG. <http://www.managementstudyguide.com/advertising-campaigns.htm>. Luettu 31.3.2017.

5 Downes, Jennifer. 2015. The Ad Ops Evolution: From Hacking to Here. Verkko-dokumentti. iProspect. <https://www.iprospect.com/en/ie/our-blog/03-15-march-ad-ops-series-1/>. Luettu 31.3.2017.

6 Strain, Mary. What Does Advertising Inventory Mean? Verkkodokumentti. Chron. <http://smallbusiness.chron.com/advertising-inventory-mean-35920.html>. Luettu 31.3.2017.

7 Koskinen, Maria. 2017. Chief Technology Officer, Relevant Partner 4 Media Oy, Helsinki. Keskustelu 6.4.2017.

8 What is Ad inventory forecasting definition? 2013. Verkkodokumentti. The digital marketing glossary. <http://www.digitalmarketing-glossary.com/What-is-Ad-inventory-forecasting-definition>. Luettu 6.4.2017.

9 How do I draw a report on a recently ended campaign? 2012. Verkkodokumentti. Cxense. <http://classroom.emediate.com/doku.php/faq:how_do_i_draw_a_report_on_a_recently_ended_campaign>. Luettu 6.4.2017.

10 Cxense Display. Verkkodokumentti. Cxense. <https://wp.cxense.com/solutions/cxense-display/>. Luettu 5.3.2017.

11 Viskari, Mikko. Tilaisitko yhden sivun web-sovelluksen (SPA) vai arkkitehtuuril-taan valmiiksi vanhentuneen järjestelmän? Verkkodokumentti. EATECH. <https://www.eatech.fi/spa-sovellukset/>. Luettu 3.2.2017.

12 Heiskanen, Henri. 2013. Yhden sivun web-sovellukset tulevat, oletko valmis? Verkkodokumentti. GOFORE. <https://gofore.com/yhden-sivun-web-sovellukset-tulevat-oletko-valmis/>. Luettu 3.2.2017.

13 HTML Introduction. Verkkodokumentti. W3Schools. <https://www.w3schools.com/html/html_intro.asp>. Luettu 5.2.2017.

14 HTML5 Introduction. Verkkodokumentti. W3Schools. <http://www.w3schools.com/html/html5_intro.asp>. Luettu 5.2.2017.

15 Segal, Nathan. An Overview of HTML5. Verkkodokumentti. QuinStreet Enterprise. <http://www.htmlgoodies.com/html5/an-overview-of-html5.html>. Luettu 5.2.2017.

16 HTML- ja CSS-opas. 2012. Verkkodokumentti. Apek. <http://salakapakka.net/oppaat/html-ja-css-opas/css_opas.php>. Luettu 5.2.2017.

17 What is JavaScript? 2015. Verkkodokumentti. Mozilla. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>. Luettu 5.2.2017.

18 JavaScript. 2017. Verkkodokumentti. Mozilla. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Luettu 5.2.2017.

19 Zhu, Henry. 2016. The State of Babel. Verkkodokumentti. Babeljs. <https://babeljs.io/blog/2016/12/07/the-state-of-babel>. Luettu 5.2.2017.

20 Shared libraries. 1999. Verkkodokumentti. IECC. <http://www.iecc.com/linker/linker09.html>. Luettu 5.2.2017.

21 Forsström, Mikko. 2015. Paras ystäväni React. Verkkodokumentti. <https://fraktio.fi/blogi/paras-ystavani-react/>. Luettu 13.2.2017.

22 React. Verkkodokumentti. Facebook. <https://facebook.github.io/react/>. Luettu 13.2.2017.

23 Thinking in React. Verkkodokumentti. Facebook. <https://facebook.github.io/react/docs/thinking-in-react.html>. Luettu 13.2.2017.

24 Three Principles. Verkkodokumentti. Redux. <http://redux.js.org/docs/introduction/ThreePrinciples.html>. Luettu 13.2.2017.

25 Bachuk, Alex. 2016. An Introduction To Redux. Verkkodokumentti. Smashing Magazine. <https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/>. Luettu 13.2.2017.

26 Duc, Pjam V. 2016. Redux - Part 1 Introduction. Verkkodokumentti. Viblo. <https://viblo.asia/phamvanduc/posts/ZjleaBBZkqJ>. Luettu 13.2.2017.

27 What are Frameworks? 22 Best Responsive CSS Frameworks for Web Design. Verkkodokumentti. AWWWARDS. <http://www.awwwards.com/what-are-frameworks-22-best-responsive-css-frameworks-for-web-design.html>. Luettu 13.2.2017.

28 Introduction. Verkkodokumentti. React-Bootstrap. < https://react-bootstrap.github.io/introduction.html>. Luettu 13.2.2017.

29 Ray, Andrew. 2016. Webpack: When To Use And Why. Verkkodokumentti. <http://blog.andrewray.me/webpack-when-to-use-and-why/>. Luettu 14.2.2017.

30 What is Webpack. Verkkodokumentti. Webpack. <https://webpack.github.io/docs/what-is-webpack.html>. Luettu 14.2.2017.

31 Renaux, Julien. 2015. Introduction to Webpack with practical examples. Verkko-dokumentti. <https://julienrenaux.fr/2015/03/30/introduction-to-webpack-with-practical-examples/>. Luettu 14.2.2017.

32 Furlott, Joseph. 2015. Get Started. Verkkodokumentti. <http://gaearon.github.io/react-hot-loader/getstarted/>. Luettu 14.2.2017.

33 Internetin rakenne ja toimintaperiaate. Verkkodokumentti. Suomen Internetopas. <http://www.internetopas.com/yleistietoa/rakenne/>. Luettu 15.2.2017.

34 About Node.js. Verkkodokumentti. Linux Foundation. <https://nodejs.org/en/about/>. Luettu 15.2.2017.

35 Salonen, Jaakko. 2012. Johdanto JavaScript-sovellusten kehitykseen Node.js:llä. Verkkodokumentti. <http://blite.iki.fi/artikkelit/javascript-nodejs-johdanto/>. Luettu 15.2.2017.

36 Marathe, Nikhil. Basics of libuv. Verkkodokumentti. <http://nikhilm.github.io/uvbook/basics.html>. Luettu 15.2.2017.

37 Hahn, Evan. 2014. Understanding Express.js. Verkkodokumentti. <https://evanhahn.com/understanding-express/>. Luettu 15.2.2017.

38 Express. Verkkodokumentti. Express. <http://expressjs.com/>. Luettu 15.2.2017.

39 About. Verkkodokumentti. The PostgreSQL Global Development Group. <https://www.postgresql.org/about/>. Luettu 16.2.2017.

40 Ekonoja, Antti; Lahtonen, Tommi & Mäntylä, Jukka. 2004. Relaatiotietokantojen peruskäsitteet. Verkkodokumentti. Jyväskylän yliopiston IT-tiedekunta ja avoin

yliopisto. <http://appro.mit.jyu.fi/doc/tiedonhallinta/tietokannat/index2.html>. Luettu 16.2.2017.

41 BCrypt. Verkkodokumentti. Assurance Technologies, LLC. <http://pythonhosted.org/passlib/lib/passlib.hash.bcrypt.html>. Luettu 16.2.2017.

42 Introduction to JSON Web Tokens. Verkkodokumentti. Auth0. <https://jwt.io/introduction/>. Luettu 16.2.2017.

43 Viljanen, Vesa. Salatut sivut. Verkkodokumentti. <https://www.yksityisyydensuoja.fi/salatut-sivut>. Luettu 16.2.2017.

44 SQL Injection. Verkkodokumentti. Acunetix. <https://www.acunetix.com/websitesecurity/sql-injection/>. Luettu 30.3.2017.

45 Schema-inspector. Verkkodokumentti. Npmjs. <https://www.npmjs.com/package/schema-inspector>. Luettu 30.3.2017.

46 Code, Manage, Collaborate. Verkkodokumentti. Bitbucket. <https://bitbucket.org/product>. Luettu 6.4.2017.

47 Kubica, Marek. 2017. Node-xml2js. Verkkodokumentti. Github. <https://github.com/Leonidas-from-XIV/node-xml2js>. Luettu 6.4.2017.